TUGAS AKHIR – KI141502

IMPLEMENTASI ADAPTIF *HELLO INTERVAL* PADA *REACTIVE ROUTING PROTOCOL AODV* BERDASARKAN TINGKAT KEPADATAN *ONE HOP VEHICLE* DI LINGKUNGAN VANET

I DEWA PUTU SUMITRA PUTRA
NRP 5113 100 009

Dosen Pembimbing I
Prof. Ir. Supeno Djanali, M.Sc., Ph. D.

Dosen Pembimbing II
Dr. Eng. Radityo Anggoro, S. Kom., M. Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

IMPLEMENTASI ADAPTIF *HELLO INTERVAL* PADA *REACTIVE ROUTING PROTOCOL AODV* BERDASARKAN TINGKAT KEPADATAN *ONE HOP VEHICLE* DI LINGKUNGAN VANET

I DEWA PUTU SUMITRA PUTRA
NRP 5113 100 009

Dosen Pembimbing I
Prof. Ir. Supeno Djanali, M.Sc., Ph. D.

Dosen Pembimbing II
Dr. Eng. Radityo Anggoro, S. Kom., M. Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION OF ADAPTIVE HELLO INTERVAL IN REACTIVE ROUTING PROTOCOL AODV BASED ON DENSITY LEVEL OF ONE HOP VEHICLE IN VANET ENVIRONMENT

I DEWA PUTU SUMITRA PUTRA
NRP 5113 100 009

Supervisor I
Prof. Ir. Supeno Djanali, M.Sc., Ph. D.

Supervisor II
Dr. Eng. Radityo Anggoro, S. Kom., M. Sc.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2017

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI ADAPTIF *HELLO INTERVAL* PADA *REACTIVE ROUTING PROTOCOL* AODV BERDASARKAN TINGKAT KEPADATAN *ONE HOP VEHICLE* DI LINGKUNGAN VANET

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
I DEWA PUTU SUMITRA PUTRA
NRP : 5113 100 009

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Prof. Ir. Supeno Djanali, M. Sc., Ph. D. .....
NIP: 19480619 197301 1 001 (pembimbing 1)

Dr. Eng. Radityo Anggoro, S. Kom., M. Sc. .....
NIP: 19841016 200812 1 002 (pembimbing 2)

KEMENTERIAN RISET, TEKNOLOGI DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI INFORMASI
DEPARTEMEN TEKNIK INFORMATIKA

SURABAYA
JULI, 2017

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI *ADAPTIVE HELLO INTERVAL* PADA *REACTIVE ROUTING PROTOCOL* AODV BERDASARKAN TINGKAT KEPADATAN *ONE HOP VEHICLE* DI LINGKUNGAN VANET

Nama Mahasiswa	**: I DEWA PUTU SUMITRA PUTRA**
NRP	**: 5113100009**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Prof. Ir. Supeno Djanali, M.Sc., Ph. D.**
Dosen Pembimbing 2	**: Dr. Eng. Radityo Anggoro, S. Kom., M. Sc**

Abstrak

AODV merupakan protokol routing yang pada dasarnya dibuat untuk digunakan pada jaringan MANET. Hal tersebut menyebabkan protokol ini tidak bekerja dengan baik pada jaringan VANET. Untuk itu diperlukan modifikasi dari protokol routing ini. Salah satu atribut yang dapat di modifikasi adalah Hello Interval. Hello Interval merupakan waktu pengiriman Hello Message dari suatu node menuju node tetangga. Node tetangga ini memiliki karakteristik mobilitas yang berbeda – beda berdasarkan kecepatan, percepatan serta jarak dengan node yang bersangkutan. Untuk lingkungan dengan mobilitas node rendah, modifikasi Hello Interval dengan nilai yang tinggi tidak begitu berpengaruh terhadap pengiriman data. Namun untuk lingkungan dengan mobilitas yang tinggi, akan muncul permasalahan yaitu kesalahan dalam pendeteksian keberadaan node tetangga.

Berdasarkan permasalahan tersebut maka akan dilakukan penelitian terhadap studi kinerja dari Hello Interval yang adaptif berdasarkan mobilitas node sekitar dengan mengadopsi metode Total Weight of Route (TWR) dan Dynamic Beacon Scheduling (DBS). Dari sana akan diketahui pengaruhnya sehingga penelitian ini bisa menjadi dasar untuk membuat AODV dengan Hello Interval yang adaptif. Modifikasi Hello Interval yang dilakukan disini disesuaikan berdasarkan keadaan mobilitas node sekitar. Penelitian ini menggunakan Network Simulator versi 3 (NS-3).

Dari uji coba yang dilakukan, Hello Interval yang adaptif memberikan peningkatan PDR hingga 10% pada skenario dengan jumlah node 50. Namun, mengalami penurunan hingga sebesar 4% - 30% pada skenario dengan jumlah node 75 dan 100.

Kata kunci : AODV, VANET, NS-3

IMPLEMENTATION OF ADAPTIVE HELLO INTERVAL IN REACTIVE ROUTING PROTOCOL AODV BASED ON DENSITY LEVEL OF ONE HOP VEHICLE IN VANET ENVIRONMENT

Student's Name	**: I DEWA PUTU SUMITRA PUTRA**
Student's ID	**: 5113100009**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Prof. Ir. Supeno Djanali, M.Sc., Ph. D.**
Second Advisor	**: Dr. Eng. Radityo Anggoro, S. Kom., M.Sc.**

Abstract

Ad Hoc On demand Distance Vector (AODV) is a routing protocol that is basically made for Mobile Ad hic Network (MANET). This causes the protocol does not work properly on the Vehicular Ad hoc Network (VANET). Thus, it required modification oh this protocol. One of the attributes that can be modified is Hello Interval. Hello Interval, the sending interval of Hello Message to the neighbor nodes which have different mobility such as speed, acceleration and distance. For a low mobility environment, Hello Interval modification do not significantly affect the performance. However, with a high mobility environment, it is possible for node to falsely detect the availability of its neighbor nodes.

Because of this problem, performance studies of adaptive Hello Interval based on neighbor nodes' mobility need to be conducted. It adopts two methods: Total Weight of Route (TWR) and Dynamic Beacon Scheduling (DBS) to determine the value of Hello Interval. From there, we will know the influenced so that this research can be the basis for making AODV with adaptive Hello Interval. The Hello Interval modifications made here are adjusted according to the circumtances of the mobility of the surrounding nodes. This research uses Network Simulator 3 (NS-3)

Based on the evaluation, adaptive Hello Interval improves its Packet Delivery Ratio for 10% at 50 nodes. However, it decrease 4% - 30% at 100 and 150 nodes.

Keyword : AODV, VANET, NS-3

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur setinggi-tingginya bagi Ida Sang Hyang Widhi Wasa, yang telah memberikan berkah dan kelancaran sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul "Implementasi Adaptif Hello Interval pada Reactive Routing Protocol AODV Berdasarkan Tingkat Kepadatan One Hop Vehicle pada VANET" dengan tepat waktu.

Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat berharga bagi penulis, karena dengan mengerjakan Tugas Akhir ini penulis dapat memperdalam, meningkatkan serta mengimplementasikan ilmu yang didapat selama penulis menempuh perkuliahan di jurusan Teknik Informatika ITS.

Terselesaikannya buku Tugas Akhir ini, tidak lepas dari bantuan dan dukungan dari berbagai pihak. Oleh karena itu, penulis ingin menyampaikan rasa terima kasih kepada:

1. Ida Sang Hyang Widhi Wasa atas berkah yang tiada habisnya sehingga penulis dapat menyelesaikan Tugas Akhir ini dengan baik.
2. Bapak dan Ibu penulis I Dewa Made Lingga dan Ni Nyoman Kerci yang telah memberikan dukungan moral dan material serta doa yang tak terhingga untuk penulis. Serta selalu memberikan semangat dan motivasi pada penulis dalam mengerjakan Tugas Akhir ini.
3. Bapak Prof. Ir. Supeno Djanali, M.Sc., Ph. D. selaku pembimbing I dan juga selaku dosen wali yang juga telah membantu, membimbing, dan memotivasi penulis dalam mengerjakan Tugas Akhir ini.
4. Bapak Dr. Eng. Radityo Anggoro, S. Kom., M. Sc. selaku pembimbing II dan juga selaku koordinator TA yang telah membantu, membimbing, dan memotivasi penulis dalam menyelesaikan Tugas Akhir ini dengan sabar.
5. Bapak Darlis Herumurti, S.Kom., M.Kom. selaku Kepala Jurusan Teknik Informatika ITS dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.

6. Teman-teman Diary Of Silver, Bagus Mayani, Kevin Arditya, Asbun, Arvi, Nyom, Sipah, Albert, Allvin, Bagas yang senantiasa menghibur dan mendukung penulis dalam mengerjakan tugas akhir ini.
7. Teman-teman TPKH angkatan 2013 yang sudah menemani bersuka cita di kampus perjuangan ini sejak maba.
8. Teman-teman TC angkatan 2013 yang sudah bersama-sama jatuh bangun menjalani kuliah di kampus TC sejak maba hingga akhir kuliah.
9. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Sebagai manusia biasa, penulis menyadari Tugas Akhir ini masih jauh dari kesempurnaan dan memiliki banyak kekurangan. Sehingga dengan segala kerendahan hari penulis mengharapkan saran dan kritik yang membangun dari pembaca.

Surabaya, Mei 2017

I Dewa Putu Sumitra Putra

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR PERSAMAAN

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dijelaskan mengenai beberapa hal dasar dalam Tugas Akhir ini yang meliputi latar belakang, perumusan masalah, batasan, tujuan dan manfaat pembuatan Tugas Akhir serta metodologi dan sistematika pembuatan buku Tugas Akhir ini. Dari uraian dibawah ini diharapkan gambaran Tugas Akhir secara umum dapat dipahami dengan baik.

1.1 Latar Belakang

Informasi merupakan suatu hal yang sangat mudah didapat dewasa ini. Setiap detiknya seseorang dapat mengetahui apa yang terjadi di belahan dunia lain. Didukung dengan perkembangan dunia internet yang sangat pesat informasi ini menjadi semakin mudah untuk didapat. Bukan hanya untuk mendapat informasi, teknologi internet saat ini juga digunakan sebagai suatu pemecahan suatu masalah. Contohnya masalah di jalan raya. Adapun beberapa contoh masalah di jalan antara lain yaitu penentuan rute untuk sampai ke tujuan lebih cepat, terjadi kecelakaan dan terjadinya kemacetan dikarenakan waktu *traffic light* disama ratakan pada suatu persimpangan. Proses penentuan rute perjalanan berhubungan dengan rute pengiriman data informasi dalam jaringan internet. Proses penentuan rute ini disebut dengan *routing*. Untuk dapat mendapatkan rute yang cepat diperlukan jalan alternatif yang banyak. Jalan alternatif disini berhubungan dengan infrastruktur yang dimana akan memakan banyak biaya untuk pembangunannya. Untuk menanggulangi hal tersebut dapat memanfaatkan teknologi jaringan *Ad-Hoc* yang mana mendasari pembuatan *Vehicular Ad-Hoc Network* (VANET).

VANET merupakan pengembangan dari *Mobile Ad-Hoc Network* (MANET) yang diaplikasikan dalam kendaraan. Jaringan VANET ini memungkinkan kendaraan saling berkomunikasi antara satu dan lainnya. Pada jaringan vanet, kendaraan dapat

berkomunikasi tanpa membutuhkan pengaturan infrastruktur tersentral ataupun server yang digunakan untuk mengontrol. Implementasi dari VANET ini telah menciptakan beberapa *routing protocol*. Berdasarkan perlakuannya terhadap rute, *routing protocol* dalam VANET dibedakan menjadi dua, yaitu *routing protocol* bersifat proaktif dan reaktif. *Routing protocol* proaktif menggunakan basis data untuk menyimpan *node* dan rute yang berada dalam jaringan, sedangkan *routing protocol* reaktif tidak menggunakan basis data melainkan melakukan pencarian *node* berikutnya pada setiap *node* untuk membentuk sebuah rute.

Tugas Akhir ini akan menggunakan *Ad hoc On-Demand Distance Vector* (AODV) yang dimodifikasi untuk meningkatkan kinerjanya. Adapun bagian yang akan di modifikasi dalam *routing protocol* ini yaitu *Interval* dari *Hello Message* yang digunakan untuk mengetahui *node* tetangga dari masing - masing *node*. *Interval* dari *Hello Message* akan diatur secara adaptif berdasarkan jumlah *node* tetangga yang ada disekitarnya.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana menentukan selang waktu *Hello Message* secara adaptif terhadap tingkat kepadatan di sekitarnya dengan mempertimbangkan faktor kecepatan, jarak dan percepatan dalam jaringan?
2. Seberapa besarkan pengaruh selang waktu *Hello Message* yang adaptif terhadap performa *routing protocol* AODV diukur dari *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO) dan *Delivery Delay*?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Routing protocol yang diuji coba adalah AODV.

2. Menentukan selang waktu Hello Message yang mengacu pada jumlah node tetangga(1-hop node) di sekitarnya.
3. Uji coba menggunakan Network Simulator 3 (NS-3).
4. Pembuatan scenario uji coba menggunakan Simulation of urban Mobility(SUMO).

1.4 Tujuan

Tujuan pengerjaan Tugas Akhir ini adalah untuk meneliti pengaruh Adaptif *Hello Interval* pada performa *routing protocol* AODV.

1.5 Manfaat

Manfaat dari pengerjaan Tugas Akhir ini antara lain :

1. Mengetahui pengaruh Adaptif *Hello Interval* pada *routing protocol* AODV.
2. Menjadi acuan untuk topik penelitian protokol reaktif yang menggunakan NS3.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.
   Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Sub bab metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir.

Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

2. Studi literatur
   Pada studi literatur ini, akan dipelajari sejumlah referensi yang diperlukan dalam pembuatan aplikasi yaitu mengenai ns-3 *Network Simulator*, *mobile ad hoc networks* (MANETs), *vehicular ad hoc networks* (VANETs), bahasa pemrograman C++, dan *routing protocol* AODV.

3. Analisis dan desain perangkat lunak
   Pada tahap ini dilakukan analisis dari hasil percobaan modifikasi *routing protocol* yang dibuat. Data yang dianalisis berasal dari perhitungan *packet delivery ratio*, *routing overhead*, dan *delay* pengiriman paket dari *node* ke *node* lainnya. Hal ini dimaksudkan untuk merumuskan solusi yang tepat untuk konfigurasi protokol reaktif AODV yang telah dimodifikasi dalam lingkungan topologi VANETs.

4. Implementasi perangkat lunak
   Implementasi yang dibuat berupa modifikasi *default class* dari protokol AODV agar bersifat adaptif terhadap kondisi topologi VANETs dengan menggunakan bahasa C++.

5. Pengujian dan evaluasi
   Setelah *routing protocol* tersebut dimodifikasi, selanjutnya dilakukan pengujian dengan Vanet *simulator generator* dan *traffic generator* untuk skenario topologi yang diujikan. Kemudian simulasi yang dibuat pada generator tersebut dijalankan pada ns-3 *Network Simulator* yang *output*-nya berupa *trace file*. Dari *trace file* tersebut akan dihitung *packet delivery ratio* (PDR), *routing overhead* (RO) dan *delivery delay* untuk mengetahui performa *routing protocol* yang telah dimodifikasi.

6. Penyusunan buku Tugas Akhir.
   Pada tahapan ini disusun buku yang memuat dokumentasi mengenai pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir secara keseluruhan. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Tinjauan Pustaka

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Desain dan Perancangan

Bab ini berisi perancangan metode yang nantunya akan diimplementasikan dan dilakukan uji coba.

Bab IV Implementasi

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa implementasi mobilitas vehicular, konfigurasi sistem dan skrip analisis yang digunakan untuk menguji performa protokol routing.

Bab V Uji Coba Dan Evaluasi

Bab ini menjelaskan tahap pengujian sistem dan pengujian performa dalam scenario mobilitas vehicular yang dibuat.

Bab VI Penutup

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

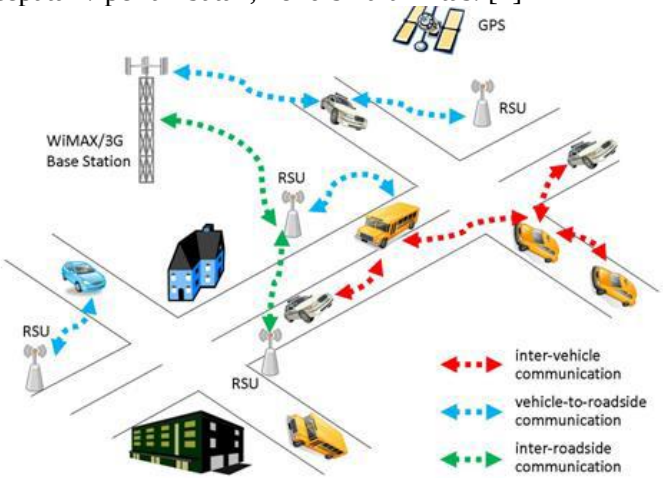
BAB II
TINJAUAN PUSTAKA

Pada bab ini akan dibahas mengenai teori yang menjadi dasar dari pembuatan Tugas Akhir ini. Teori yang dibahas mencakup elemen-elemen yang terkait dalam topik Tugas Akhir mulai dari sumber dari permasalahan, pendekatan yang digunakan, serta metode dan teknologi yang digunakan untuk pengerjaan Tugas Akhir ini.

2.1 *Vehicular Ad hoc Network*

Sebuah jaringan terorganisir yang dibentuk dengan menghubungkan kendaraan dan RSU (*Roadside Unit*) disebut *Vehicular Ad Hoc Network* (VANET), dan RSU lebih lanjut terhubung ke jaringan *backbone* berkecepatan tinggi melalui koneksi jaringan. Kepentingan peningkatan baru-baru ini telah diajukan pada aplikasi melalui V2V (*Vehicle to Vehicle*) dan V2I (*Vehicle to Infrastructure*) komunikasi, bertujuan untuk meningkatkan keselamatan mengemudi dan manajemen lalu lintas sementara menyediakan *driver* dan penumpang dengan akses Internet. Dalam VANETs, RSUs dapat memberikan bantuan dalam menemukan fasilitas seperti restoran dan pompa bensin, dan membroadcast pesan yang terkait seperti (maksimum kurva kecepatan) pemberitahuan untuk memberikan pengendara informasi. Sebagai contoh, sebuah kendaraan dapat berkomunikasi dengan lampu lalu lintas cahaya melalui V2I komunikasi, dan lampu lalu lintas dapat menunjukkan ke kendaraan ketika keadaan lampu ke kuning atau merah. Ini dapat berfungsi sebagai tanda pemberitahuan kepada pengemudi, dan akan sangat membantu para pengendara ketika mereka sedang berkendara selama kondisi cuaca musim dingin atau di daerah asing. Hal ini dapat mengurangi terjadinya kecelakaan. Melalui komunikasi V2V, pengendara bisa mendapatkan informasi yang lebih baik dan mengambil tindakan

awal untuk menanggapi situasi yang abnormal. Untuk mencapai hal ini, suatu OBU secara teratur menyiarkan pesan yang terkait dengan informasi dari posisi pengendara, waktu saat ini, arah mengemudi, kecepatan, status rem, sudut kemudi, lampu sen, percepatan / perlambatan, kondisi lalu lintas. [1]



Gambar 2.1 Ilustrasi Vanet [2]

Pada tugas akhir ini jaringan yang digunakan adalahh nirkabel dan protokol yang digunakan adalah AODV.

2.2 Protokol *Routing* AODV (*Ad hoc On Demand Distance Vector*)

Ad hoc On-Demand Distance Vector (AODV) adalah *distance vector routing protocol* yang termasuk dalam klasifikasi *reactive routing protocol*, yang hanya melakukan *request* sebuah rute saat akan mengirimkan paket. AODV yang standar ini dikembangkan oleh C. E. Perkins, E.M. Belding-Royer dan S.Das pada RFC 3561. Ciri utama dari AODV adalah menjaga *timer-based state* pada setiap node sesuai dengan penggunaan tabel routing. Tabel routing akan kadaluarsa jika jarang digunakan. AODV memiliki *route discovery* dan *route maintenance*. *Route*

Discovery berupa *Route Request* (RREQ) dan *Route Reply* (RREP). Sedangkan *Route Maintenance* berupa *Data*, *Route update* dan *Route Error* (RERR).

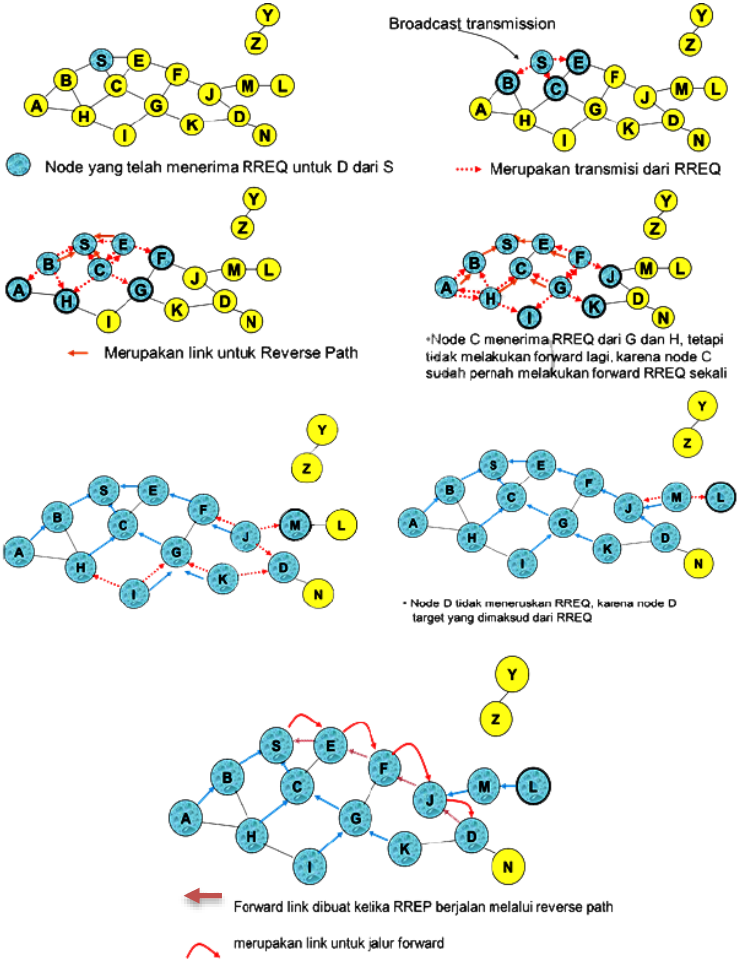
AODV memerlukan beberapa *field* pada setiap node untuk menjaga tabel routing, antara lain :

- *Destination IP Address* : berisi alamat IP dari node tujuan yang digunakan untuk menentukan rute
- *Destination Sequence Number* : *destination sequence number* bekerjasama untuk menentukan rute
- *Next Hop* : 'Loncatan' (*hop*) berikutnya, bisa berupa tujuan atau node tengah, *field* ini dirancang untuk meneruskan paket ke node tujuan.
- *Hop Count* : Jumlah *hop* dari alamat IP sumber ke alamat IP tujuan.
- *Lifetime* : Waktu dalam *miliseconds* yang digunakan untuk node menerima RREP.
- *Routing Flags* : Status sebuah rute, *up* (valid), *down* (tidak valid) atau sedang diperbaiki.

Penemuan jalur (*Path discovery*) atau *Route discovery* pada AODV diinisiasi dengan menyebarkan *Route Reply* (RREP). Ketika RREP menjelajahi node RREP akan secara otomatis melakukan *setup path*. Jika sebuah *node* menerima RREP, maka *node* tersebut akan mengirimkan RREP lagi ke *node sequence number*. Pada proses ini, *node* pertama kali akan mengecek *destination sequence number* pada tabel routing, apakah lebih besar dari 1 (satu) pada *Route Request* (RREQ), jika benar, maka *node* akan mengirim RREP. Ketika RREP berjalan kembali ke *source* melalui *path* yang telah dilakukan saat *setup* sebelumnya, RREP akan melakukan *setup* jalur ke depan dan melakukan *update timeout*.

Jika sebuah *link* ke *hop* berikutnya tidak dapat dideteksi dengan metode penemuan rute, maka link tersebut akan diasumsikan putus dan *Route Error* (RERR) akan disebarkan ke node tetangganya. Dengan demikian sebuah node bisa menghentikan pengiriman data melalui rute ini atau meminta rute

baru dengan menyebarkan RREQ kembali [3]. Ilustrasi teknik pencarian rute pada AODV dapat dilihat pada Gambar 2.2. Pada tugas akhir ini protokol AODV digunakan sebagai protokol VANET.



Gambar 2.2 Teknik pencarian rute dari AODV [17]

2.3 *Total Weigth of Route* (TWR)

Pada Tugas Akhir ini, modifikasi *Hello Interval* dilakukan dengan mempertimbangkan faktor kecepatan, jarak dan juga percepatan. Oleh karena itu diperlukan suatu formulasi untuk menggabungkan parameter-parameter tersebut. *Total Weigth of Route* (TWR) merupakan bobot sebuah route dinilai dari kecepatan, percepatan dan juga jarak *node* ke *node* berikutnya [4].

Formula TWR dirancang dengan mempertimbangkan beberapa faktor berikut :

- Kecepatan dan Percepatan

  Semakin besar perbedaan kecepatan dan akselerasi antar dua kendaraan, maka semakin besar pula kemungkinan dua kendaraan tersebut saling berjauhan. Asumsi tersebut berimplikasi pada TWR dalam bentuk pelebaran nilai TWR. Alasannya adalah jika kedua kendaraan saling berjauhan, maka semakin besar kemungkinan terjadinya kerusakan koneksi antar kendaraan tersebut. Jika kedua kendaraan tersebut berada di dalam radius komunikasi dan bergerak dengan kecepatan dan akselerasi yang relatif sama antara satu dengan yang lainnya, maka bisa diasumsikan kedua kendaraan tersebut akan berada pada radius komunikasi dalam durasi yang lebih lama.
- Arah kendaraan

  Jika dua kendaraan yang bergerak dengan arah yang relatif sama, maka kedua kendaraan tersebut akan berada dalam radius komunikasi dalam durasi yang lebih lama.
- Kualitas hubungan antar kendaraan

  Kualitas hubungan yang dimaksud adalah apakah *node* yang mengirimkan paket dan *node* penerimanya masih berada dalam radius komunikasi. Dalam lingkungan VANET, terdapat banyak *neighbor node*, bangunan dan halangan lainnya yang dapat mempengaruhi kualitas hubungan. Kualitas hubungan didefinisikan dalam indeks stabilitas (*stability index*) [4]. Formula dari indeks stabilitas dapat dilihat pada Persamaan 2.1.

$$s_{ij} = 1 - \frac{\min\left(\sqrt{(i_x - j_x)^2 + \left(i_y - j_y\right)^2}\,; r\right)}{r} \quad (2.1)$$

dimana,
i_x, i_y : Koordinat dari kendaraan i
j_x, j_y : Koordinat dari kendaraan j
r : Jarak transmisi maksimum

Nilai indeks stabilitas yang paling baik adalah 1. Indeks stabilitas dengan nilai 1 dihasilkan ketika kendaraan i dan j memiliki vektor pergerakan yang sama. Nilai indeks stabilitas paling buruk adalah 0. Indeks stabilitas dengan nilai 0 didapatkan ketika kendaraan i dan j memiliki jarak yang lebih jauh dari jarak transmisi maksimum. Indeks stabilitas dengan nilai 0 mengimplikasikan kendaraan i dan j berada di luar radius komunikasi dari masing – masing kendaraan. Kemudian nilai kualitas hubungan Q [4] dihitung dari rumus pada Persamaan 2.2.

$$Q = \frac{1}{s_{ij}} \quad (2.2)$$

Berdasarkan faktor – faktor di atas, rumus TWR [4] di ekspresikan dengan Persamaan 2.3.

$$TWR = f_s \times |S_n - S_d| + f_a \times |A_n - A_d| + f_d \times |\theta_n - \theta_d| + f_q \times Q \quad (2.3)$$

dimana,
S_n, A_n, θ_n : Kecepatan, akselerasi dan arah *next-hop node*
S_d, A_d, θ_d : Kecepatan, akselerasi dan arah *node* tujuan
f_s : Faktor pengali kecepatan
f_a : Faktor pengali akselerasi
f_d : Faktor pengali arah
f_q : Faktor pengali kualitas hubungan
Q : Kualitas hubungan antara *node* sumber dengan *next-hop node*.

Nilai TWR ditentukan oleh perbedaan kecepatan, akselerasi dan arah dari *next-hop node* dan *node* tujuan, serta kualitas hubungan antara *node* sumber dengan *next-hop node*. Formula TWR mengimplikasikan *next-hop node* yang baik adalah node yang memiliki nilai TWR yang rendah karena memiliki kecepatan, akselerasi dan arah yang relatif sama dengan *node* tujuan, serta memiliki kualitas hubungan yang relatif bagus dengan *node* sumber [4].

2.4 *Dynamic Beacon Scheduling*

Disini akan diilustrasikan prosedur dari penentuan nilai beacon secara dinamis untuk menghindari *Routing Overhead*. Penentuan nilai *interval* (β) ditentukan berdasarkan nilai kecepatan masing-masing kendaraan karena kecepatan bisa dianggap sebagai tingkat dimana kendaraan menempuh jarak. Sebuah kendaraan dengan kecepatan tinggi cenderung menempuh jarak yang jauh, berbeda dengan kendaraan berkecepatan rendah yang cenderung menempuh jarak lebih pendek dengan waktu tempuh yang sama. Jadi, kemungkinan *node* tetangga hilang pada kendaraan berkecepatan tinggi adalah lebih sering dari pada kendaraan berkecepatan rendah. Maka dari itu semakin cepat kendaran, semakin sering mengirimkan *Hello Message*. Solusi yang ditawarkan menggunakan empat parameter yaitu : kecepatan rendah (S_{slow}), kecepatan tinggi (S_{fast}), minimum interval (β_{min}), dan maximum interval (β_{max}). Perhitungan *beacon* dapat dilihat pada Persamaan 2.4.

$$\beta_i = \begin{cases} \beta_{min} & if\ S_i \geq S_{fast} \\ \frac{S_{fast}}{S_i} * \beta_{min} & if\ S_{slow} \leq S_i \leq S_{fast} \\ \beta_{max} & if\ S_i \leq S_{slow} \end{cases} \quad (2.4)$$

Minimum beacon interval dan *maximum beacon interval* dibedakan untuk setiap pengaplikasian dan di variasikan terhadap kebutuhan komunikasi [5]. Pada Tugas Akhir ini rumus diatas

akan dipakai sebagai acuan penentuan *hello interval* dengan mengganti faktor *speed* dengan faktor TWR.

2.5 *Simulation of Urban Mobility* (SUMO)

Simulation of Urban Mobility atau dikenal dengan SUMO adalah program aplikasi simulator yang digunakan untuk membuat simulasi pergerakan kendaraan pada suatu jalur tertentu. SUMO adalah program yang bersigat *free*, *open-source*, berukuran kecil, dan simulasi trafik multi-modal. SUMO dikembangkan pada tahun 2000 yang bertujuan untuk mengakomodasi penelitian-penelitian yang melibatkan pergerakan kendaraan di jalan raya, terutama daerah-daerah yang padat penduduknya. SUMO dikembangkan kali pertama oleh Daniel Krajzewicz, Eric Nikolay, dan Michael Behrisch [6]

SUMO terdiri dari banyak *tools* yang memiliki fungsi berbeda-beda. Berikut merupakan penjelasan dari fungsi tools yang digunakan pada pembuatan Tugas Akhir ini :

- netgenerate.exe
  Netgenerate merupakan *tools* yang berfungsi untuk membuat peta seperti grid, *spider* dan bahkan *random network*. Netgenerate juga berfungsi untuk menentukan kecepatan maksimum jalan dan membuat *traffic light* pada peta. Hasil dari netgenerate ini berupa file dengan ekstensi .net.xml. Pada Tugas Akhir ini netgenerate digunakan untuk membuat peta untuk scenario grid.
- netconvert.exe
  Netconvert merupakan *tools* yang digunakan untuk mengkonversi peta yang didapat dari OpenStreetMap yang berekstensi .osm agar sesuai dengan format dari SUMO yaitu .net.xml. Pada Tugas Akhir ini netconvert digunakan untuk mengkonversi peta daerah Surabaya dari OpenStreetMap ke dalam SUMO.
- randomTrips.py
  randomTrips.py merupakan *tools* pada SUMO untuk membuat *node* beserta titik awal dan tujuannya secara acak. Hasil dari

randomTrips.py adalah berupa *file* dengan ekstensi .trips.xml. Pada Tugas Akhir ini randomTrips.py digunakan untuk membuat *node* pada scenario grid dan Surabaya.

- duarouter.exe
  Duarouter berfungsi untuk membuat rute pergerakan yang asal dan tujuannya sudah dibuat oleh randomTrips.py. Secara *default* duarouter menggunakan algoritma Djikstra untuk membuat rutenya. Hasil dari duarouter berupa file dengan ekstensi .rou.xml
- sumo-gui.exe
  Sumo-gui berfungsi untuk menampilkan pergerakan setiap *node* yang sudah dibuat sebelumnya secara grafis. Cara kerja sumo-gui adalah dengan menggabungkan peta yang ada pada *file* berekstensi .net.xml dengan rutenya yang berekstensi .rou.xml dalam sebuah *file* lain dengan format .sumocfg.
- sumo.exe
  Sumo.exe berfungsi untuk membuat *file* berekstensi .xml yang kemudian dapat diekspor ke dalam berbagai bahasia simulasi, salah satunya NS-3 dengan tools traceExporter.py.
- traceExporter.py
  traceExporter.py digunakan untuk mengekspor *file* hasil SUMO yang berekstensi .xml agar bisa digunakan pada NS-3 dengan menggunakan fungsi NS2mobility-output pada traceExporter.

2.6 *VMware Workstation*

VMware Workstation adalah industri untuk menjalankan beberapa sistem operasi sebagai mesin virtual pada satu PC. Ribuan profesial dan pengembang IT menggunakannya untuk lebih produktif. Produk *workstation* memungkinkan pengguna menjalankan sistem operasi termasuk Linux, Windows dan lainnya sebagai mesin virtual. Pengguna dapat meniru lingkungan *server*, *desktop* dan tablet pada mesin virtual, untuk menjalankan aplikasi

secara bersamaan di seluruh sistem operasi tanpa melakukan *booting* ulang.

Produk *workstation* memudahkan hampir seluruh sistem operasi untuk melakukan testing pada *platform* mereka. Mulai dari testing aplikasi yang dibangun untuk Windows 10, testing berbagai browser sampai dengan *deployment* Android-x86 untuk perangkat bergerak tanpa memerlukan perangkatnya tersendiri. Professional IT menggunakan produk *workstation* untuk terhubung dengan aman ke *vSphere*, *ESXi* atau *server Workstation* lainnya untuk mengelola mesin virtual dan *host* fisik. *Platform hypervisor* yang umum memaksimalkan ketangkasan dan produktivitas dengan memungkinkan pengalihan mesin virtual ke dan dari PC local anda dengan mudah [7].

Pada tugas akhir ini *VMware Workstation* digunakan untuk menjalankan sistem operasi Linux yang berisikan NS-3 dan sebagai penghubung antara sistem operasi Windows dan Linux.

2.7 *OpenStreetMap*

OpenStreetMap adalah proyek bebas yang mengumpulkan data spasial dan dapat digunakan secara bebas (*Open Data*). Data tersebut digunakan untuk membangun peta dunia dan peta-peta khusus yang diturunkan dan dimanfaatkan untuk beragam kebutuhan termasuk navigasi. OpenStreetMap memungkinkan siapa saja untuk melihat, mengedit dan menggunakan data geografis yang telah dibangun secara kolaboratif dari mana dan oleh siapa saja di permukaan bumi ini.

Inti dari proyek ini adalah berupa sebuah *database* geografis mirip Wiki yang dapat dimanfaatkan secara bebas berdasarkan Lisensi *Open Database*. Dengan demikian, pemanfaatan untuk keperluan cetak, *website* dan untuk aplikasi perangkat lunak seperti navigasi tidak dibatasi maupun ditarik biaya, asalkan menyebutkan *OpenStreetMap* sebagai sumber data. *OpenStreetMap* tidak lain merupakan *Wikipedia* untuk data peta dunia [8].

Pada tugas akhir ini *OpenStreetMap* digunakan sebagai penyedia peta untuk skenario riil Surabaya.

2.8 JOSM

JOSM(*Java OpenStreetMap Editor*) adalah alat penyunting data *OpenStreetMap*. JOSM tidak membutuhkan koneksi internet untuk menyunting dan cocok bagi pengguna yang sudah mahir. JOSM awalnya dikembangkan oleh Immanuel Scholz dan saat ini dikelola oleh Dirk Sticker. Homepagenya terletak di josm.openstreetmap.de. Java harus terpasang terlebih dahulu sebelum dapat mengoperasikan JOSM [9].

Pada tugas akhir ini JOSM digunakan untuk mengedit peta yang didapat dari *OpenStreetMap.*

2.9 AWK

Awk merupakan bahasa pemrograman yang digunakan untuk manipulasi data dan membuat laporan. AWK adalah sebuah program *filter* untuk teks, seperti halnya perintah "grep". Awk dapat digunakan untuk mencari bentuk atau model dalam sebuah *file* teks. Awk juga dapat mengganti bentuk satu teks ke dalam bentuk teks yang lain. Awk dapat juga digunakan untuk melakukan proses aritmatikla seperti yang dilakukan oleh perintah "expr". Awk adalah sebuah pemrograman seperti pada *shell* atau C yang memiliki karakteristik yaitu sebagai *tool* yang cocok untuk *jobs* juga sebagai pelengkap (*complicated*) untuk *filter* standar [10].
Pada tugas akhir ini AWK digunakan untuk membuat *script* perhitungan PDR, *delay*, dan *routing overhead* dari hasil trace NS-3.

2.10 *Network Simulator* 3 (NS-3)

Simulator NS-3 adalah sebuah *network simulator* yang memiliki ciri tersendiri yang ditargetkan secara utama untuk tujuan riset dan pendidikan. Proyek NS-3, dimulai pada 2006, adalah sebuah proyek *open source* yang diatur oleh komunitas peneliti dan

pengembang. NS-3 utamanya digunakan pada sistem operasi Linux, namun dapat digunakan untuk *FreeBSD*, *Cygwin* (Untuk Windows), dan dukungan untuk *Visual Studio* yang bersifat *native* yang masih dikembangkan.

Pada VANET, NS-3 mempunyai beberapa fitur yang dapat dimanfaatkan untuk memodelkan dan menguji VANET. VANET disimulasikan dengan membuat skenario jaringan dan penerapan *routing protocol*. Pada modul NS-3 versi 3.26 terdapat *source* untuk melakukan percobaan lingkungan VANET. Pembuatan topologi, node dan protokol yang digunakan untuk VANET sudah didukung oleh NS-3. Skenario *mobility* dengan format NS-2 juga dapat digunakan pada NS-3 dengan bantuan *library* atau *helper* yang telah tersedia. Dengan NS-3 kita dapat menambahkan fungsi-fungsi baru di dalam *core* NS-3 karena NS-3 bersifat *open source*. NS-3 dikembangkan menggunakan Bahasa C++ di lapisan inti dan *script python*. Fitur-fitur NS-3 diantaranya adalah sistem atribut NS-3 terdokumentasi dengan baik. Setiap objek NS-3 memiliki seperangkat atribut (*name, type, initial value*) dan NS-3 selaras dengan sistem nyata.

Simulator NS-3 menyediakan berbagai banyak modul yang mendukung berbagai macam simulasi skenario. Simulasi yang efektif untuk *Vehicular Ad hoc Network*(VANET) membutuhkan model yang bisa menangkap jaringan *wireless* yang sangat dinamis, propagasi dan karakter mobilitas dari skenario. NS-3 juga terintegrasi dengan *software* atau *tools* lain seperti *wireshark* untuk melihat *trace output*. Representasi hasil data simulasi NS-3 dapat ditampilkan dalam bentuk grafik, sehingga memudahkan untuk menganalisa dan mengevaluasi hasil terhadap suatu model jaringan VANET [11].

Pada tugas akhir ini NS-3 digunakan untuk mensimulasikan scenario VANET.

2.10.1 Instalasi NS-3

Sebelum melakukan instalasi NS-3 pastikan pada perangkat yang akan dipasang sudah memiliki RAM lebih besar dari 1 GB, karena ada kemungkinan *out of memory* saat instalasi

menggunakan RAM 1 GB. Petunjuk instalasi NS-3 selengkapnya akan dijelaskan pada lampiran 2.

2.10.2 Penggunaan vanet-routing-compare.cc

Pada NS-3, terdapat skrip program untuk VANET yaitu vanet-routing-compare yang terletak pada direktori src/wave/examples. Program ini mengkombinasikan beberapa modul NS-3 dari *examples* untuk mendukung pengembangan simulasi VANET.

Program vanet-routing-compare memiliki berbagai jenis parameter simulasi yang dapat dikonfigurasi sesuai kebutuhan. Kategori parameter yang dapat dirubah nilai aslinya dan kegunaannya serta contoh cara penggunaannya serta konfigurasinya akan ditunjukkan oleh Tabel 2.1.

Tabel 2.1 Parameter vanet-routing-compare

Parameter	Meaning	Values	Default	Example
totaltime	Simulation end time	Double	300.01	./waf --run "scratch/my-vanet-routing-compare --totaltime=200"
nodes	Number of nodes (i.e. vehicles)	integer	156	./waf --run "scratch/my-vanet-routing-compare --nodes=50"
sinks	Number of routing sinks	integer	10	./waf --run "scratch/my-vanet-routing-compare --sinks=1"
txp	Transmit power (dB), e.g. txp=7.5	Double	7.5	./waf --run "scratch/my-vanet-routing-compare --txp=7"
protocol	Protocol that used	Integer	2	./waf --run "scratch/my-vanet-

Parameter	Meaning	Values	Default	Example
	in simulation 1=OLSR 2=AODV 3=DSDV 4=DSR			routing-compare --protocol=2”
lossModel	1=Friis 2=ItuR1411Los 3=TwoRayGround 4=LongDistance	integer	3	./waf --run “scratch/my-vanet-routing-compare --lossModel=3”
fading	0=None 1=Nakagami(buildings=1 overrides)	integer	0	./waf --run “scratch/my-vanet-routing-compare --fading=1”
phyMode	Wifi Phy Mode	String	OfdnRate6MbpsBW10MHz	./waf --run “scratch/my-vanet-routing-compare --phyMode= OfdnRate6MbpsBW10MHz”
80211Mode	1=802.11p 2=802.11b 3=WAVE-PHY	integer	1	./waf --run “scratch/my-vanet-routing-compare --80211Mode=1”
traceFile	Ns2 movement trace file	String	/src/wave/example/low_ct-unterstrass-1day/filt	./waf --run “scratch/my-vanet-routing-compare --traceFile=/scratch/grid1/tcl”
mobility	1=trace 2=RWP	integer	1	./waf --run “scratch/my-vanet-

Parameter	Meaning	Values	Default	Example
				routing-compare --mobility=1”
rate	Rate	String	512bps	./waf --run “scratch/my-vanet-routing-compare --rate=2048”
phyModeB	Phy mode 802.11p	String	DsssRate 11Mbps	./waf --run “scratch/my-vanet-routing-compare --phyModeB=DsssRate11Mbps”
speed	Node speed (m/s)	integer	20	./waf --run “scratch/my-vanet-routing-compare --speed=10”
pause	Node pause(sp)	integer	0	./waf --run “scratch/my-vanet-routing-compare --pause=0”
verbose	0=quite 1=verdose	integer	0	./waf --run “scratch/my-vanet-routing-compare --verbose=1”
scenario	1=synthetic 2=playback-trace	integer	1	./waf --run “scratch/my-vanet-routing-compare --scenario=3”
gpsaccuracy	GPS time accuracy, in ns	Double	40	./waf --run “scratch/my-vanet-routing-compare --gpsaccuracy=40”
txmaxdelay	Tx max delay , in ms	Double	10	./waf --run “scratch/my-vanet-routing-compare --txmaxdelay=1”
routingTables	Dump routing	integer	0	./waf --run “scratch/my-vanet-

Parameter	Meaning	Values	Default	Example
	table at t=5			routing-compare --routingTables=1”
asciiTrace	Dump ASCII trace data	integer	0	./waf --run “scratch/my-vanet-routing-compare --asciiTrace=1”
pcap	Create PCAP files for all nodes	integer	0	./waf --run “scratch/my-vanet-routing-compare --pcap=1”

2.10.3 NS-3 *Trace File*

NS-3 *Trace File* merupakan *output* hasil *running* skenario dari NS-3 yang biasanya berekstensi .tr. Dari *trace file* inilah yang kemudian akan dihitung PDR, *Average Delivery Delay* dan *Routing Overhead* dari protokol yang digunakan. NS-3 *Trace File* berisi *log* tentang semua jenis pengiriman yang terjadi selama. Di dalam NS-3 *Trace File* ini terdapat berbagai jenis pengiriman paket antara lain, *Hello Message*, data paket, RREQ (*route request*), RREP(*route reply*), dan RRER (*route error*).

Gambar 2.3 merupakan contoh pengiriman *Hello Message* pada NS-3 *Trace File*.

```
t 0.006
/NodeList/19/DeviceList/0/$ns3::WifiNetDevice/Phy/State/Tx
ns3::WifiMacHeader (DATA ToDS=0, FromDS=0, MoreFrag=0,
Retry=0, MoreData=0 Duration/ID=0us, DA=ff:ff:ff:ff:ff:ff,
SA=00:00:00:00:00:14, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=0) ns3::LlcSnapHeader (type 0x800)
ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 1 id
0 protocol 17 offset (bytes) 0 flags [none] length: 48
10.1.0.20 > 10.1.255.255) ns3::UdpHeader (length: 28 654 >
654) ns3::aodv::TypeHeader (RREP) ns3::aodv::RrepHeader
(destination: ipv4 10.1.0.20 sequence number 0 source ipv4
10.1.0.20 lifetime 200 acknowledgment required flag 0)
ns3::WifiMacTrailer ()
```

Gambar 2.3 *Trace* pengiriman *Hello Message*

Hello Message bisa dibilang merupakan jenis RREP special karena *TypeHeader*-nya merupakan RREP namun yang membedakannya adalah tujuan pengirimannya. Dalam contoh tersebut ip tujuannya merupakan sebuah ip *broadcast* (10.1.255.255) dengan *destination* dan *source* ip yang merupakan ip *node* itu sendiri (10.1.0.20). Saat melakukan simulasi VANET menggunakan AODV, baris yang paling sering muncul pasti *Hello Message*, baik pengiriman maupun penerimaan. Karena *Hello Message* pasti dikirimkan secara periodik meskipun tidak ada aktivitas pengiriman data paket.

Gambar 2.4 merupakan contoh penerimaan *Hello Message* pada NS-3 *Trace File*.

```
r 0.00616045
/NodeList/2/DeviceList/0/$ns3::WifiNetDevice/Phy/State/RxO
k ns3::WifiMacHeader (DATA ToDS=0, FromDS=0, MoreFrag=0,
Retry=0, MoreData=0 Duration/ID=0us, DA=ff:ff:ff:ff:ff:ff,
SA=00:00:00:00:00:14, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=0) ns3::LlcSnapHeader (type 0x800)
ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 1 id
0 protocol 17 offset (bytes) 0 flags [none] length: 48
10.1.0.20 > 10.1.255.255) ns3::UdpHeader (length: 28 654 >
654) ns3::aodv::TypeHeader (RREP) ns3::aodv::RrepHeader
(destination: ipv4 10.1.0.20 sequence number 0 source ipv4
10.1.0.20 lifetime 200 acknowledgment required flag 0)
ns3::WifiMacTrailer ()
```

Gambar 2.4 *Trace* penerimaan *Hello Message*

Yang membedakan pengiriman *Hello Message* dengan penerimaan *Hello Message* adalah pada kolom pertama. Pengiriman *Hello Message* diawali dengan “t” yang berarti *transmit*, sedangkan penerimaan *Hello Message* diawali dengan “r” yang berarti *received*. Biasanya satu pengiriman *Hello Message* akan diterima oleh beberapa *node* karena sifatnya *broadcast message*.

Gambar 2.5 merupakan contoh pengiriman data paket pada NS-3 *Trace File*.

```
t 8.2038
/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/State/Tx
ns3::WifiMacHeader (DATA ToDS=0, FromDS=0, MoreFrag=0,
Retry=0, MoreData=0 Duration/ID=96us,
DA=00:00:00:00:00:15, SA=00:00:00:00:00:02,
BSSID=ff:ff:ff:ff:ff:ff, FragNumber=0, SeqNumber=87)
ns3::LlcSnapHeader (type 0x800) ns3::Ipv4Header (tos 0x0
DSCP Default ECN Not-ECT ttl 64 id 0 protocol 17 offset
(bytes) 0 flags [none] length: 92 10.1.0.2 > 10.1.0.1)
ns3::UdpHeader (length: 72 49153 > 9) Payload (size=64)
ns3::WifiMacTrailer ()
```

Gambar 2.5 *Trace* pengiriman data paket

Pengiriman data paket pasti memiliki pengirim dan penerima yang jelas dan selalu sama. Dalam skenario yang akan di jalankan *node* 1 dengan IP 10.1.0.2 akan bertindak sebagai pengirim dan *node* 0 dengan IP 10.1.0.1 akan bertindak sebagai penerima. Data paket memiliki ciri-ciri terdapat *Payload* (size=64) yang merupakan ukuran data yang dikirim. Selain itu data paket memiliki *id* paket yang akan berguna saat menghitung PDR dan *average delivery delay*. Pada contoh yang diberikan, terlihat *id* paket adalah 0 yang berarti ini merupakan paket pertama.

Gambar 2.6 merupakan contoh penerimaan data paket pada NS-3 *Trace File*.

```
r 8.69356
/NodeList/46/DeviceList/0/$ns3::WifiNetDevice/Phy/State/Rx
Ok ns3::WifiMacHeader (DATA ToDS=0, FromDS=0, MoreFrag=0,
Retry=0, MoreData=0 Duration/ID=96us,
DA=00:00:00:00:00:18, SA=00:00:00:00:00:0b,
BSSID=ff:ff:ff:ff:ff:ff, FragNumber=0, SeqNumber=185)
ns3::LlcSnapHeader (type 0x800) ns3::Ipv4Header (tos 0x0
DSCP Default ECN Not-ECT ttl 62 id 27 protocol 17 offset
(bytes) 0 flags [none] length: 92 10.1.0.2 > 10.1.0.1)
ns3::UdpHeader (length: 72 49153 > 9) Payload (size=64)
ns3::WifiMacTrailer ()
```

Gambar 2.6 *Trace* penerimaan data paket

Sama seperti *Hello Message*, yang membedakan paling jelas antara pengiriman dan penerimaan adalah perbedaan “t” dan “r” pada kolom pertama. Setelah menerima data paket, *node* yang menerima harus mengirim ACK agar pengirim tahu bahwa yang dikirim sudah sampai.

Gambar 2.7 merupakan contoh pengiriman RREQ (*route request*) pada NS-3 *Trace File*.

```
t 1.94742
/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/State/Tx
ns3::WifiMacHeader (DATA ToDS=0, FromDS=0, MoreFrag=0,
Retry=0, MoreData=0 Duration/ID=0us, DA=ff:ff:ff:ff:ff:ff,
SA=00:00:00:00:00:02, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=20) ns3::LlcSnapHeader (type
0x800) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT
ttl 1 id 20 protocol 17 offset (bytes) 0 flags [none]
length: 52 10.1.0.2 > 10.1.255.255) ns3::UdpHeader
(length: 32 654 > 654) ns3::aodv::TypeHeader (RREQ)
ns3::aodv::RreqHeader (RREQ ID 1 destination: ipv4
10.1.0.1 sequence number 0 source: ipv4 10.1.0.2 sequence
number 1 flags: Gratuitous RREP 1 Destination only 0
Unknown sequence number 1) ns3::WifiMacTrailer ()
```

Gambar 2.7 *Trace* pengiriman *Route Request* (RREQ)

Route Request memiliki ciri utama yaitu memiliki ns3::aodv::TypeHeader (RREQ). Berisi informasi *destination* ip dan *source* ip dan dikirimkan ke semua *node* tetangga secara *broadcast* sampai diterima oleh *node* tujuan. *Route Request* (RREQ) yang akan dihitung nantinya adalah yang memiliki *destination*: ipv4 10.1.0.1 dan *source*: ipv4 10.1.0.2. Ciri lain dari *Route Request* (RREQ) yaitu tujuan pengiriman pasti alamat IP 10.1.255.255 atau *broadcast*, karena *Route Request* (RREQ) bertujuan untuk mencari tahu letak *node* yang dicari sehingga tidak akan berhenti melakukan *broadcast* hingga node yang dituju menerima *Route Request* tersebut dan membalasnya dengan mengirimkan *Route Reply* (RREP).

Gambar 2.8 merupakan contoh pengiriman RREP (*Route Reply*) pada NS-3 *Trace File*.

```
r 30.00616045
/NodeList/2/DeviceList/0/$ns3::WifiNetDevice/Phy/State/
RxOk ns3::WifiMacHeader (DATA ToDS=0, FromDS=0,
MoreFrag=0, Retry=0, MoreData=0 Duration/ID=0us,
DA=ff:ff:ff:ff:ff:ff, SA=00:00:00:00:00:14,
BSSID=ff:ff:ff:ff:ff:ff, FragNumber=0, SeqNumber=0)
ns3::LlcSnapHeader (type 0x800) ns3::Ipv4Header (tos
0x0 DSCP Default ECN Not-ECT ttl 1 id 0 protocol 17
offset (bytes) 0 flags [none] length: 48 10.1.0.20 >
10.1.0.19) ns3::UdpHeader (length: 28 654 > 654)
ns3::aodv::TypeHeader (RREP) ns3::aodv::RrepHeader
(destination: ipv4 10.1.0.20 sequence number 0 source
ipv4 10.1.0.20 lifetime 200 acknowledgment required
flag 0) ns3::WifiMacTrailer ()
```

Gambar 2.8 *Trace* pengiriman *Route Reply* (RREP)

Route reply mirip seperti *Hello Message* namun tujuan pengirimannya bukanlah IP *broadcast* (10.1.255.255) melainkan suatu *node* tertentu. Selain itu *destination* dan *source* IP tidak bernilai sama seperti pada *Hello Message*. Pada *route reply* juga diperlukan ACK setelah menerima RREP. *Route Reply* (RREP). Pada awalnya akan dikirim oleh *node* tujuan yaitu *node* 0, kemudian ACK akan dikirimkan oleh node pengirim RREQ kepada *node* 0 secara bertahap sehingga sampai kepada *Node* yang pertama kali mengirimkan RREQ yaitu *node* 1. *Node-node* yang menjadi pengirim RREP hingga mencapai *node* 1 akan menjadi *hop* dalam pengiriman data paket. Gambar 2.9 merupakan contoh pengiriman RRER (*Route Error*) pada NS-3 *Trace File*.

```
t 23.00616045
/NodeList/2/DeviceList/0/$ns3::WifiNetDevice/Phy/State/
RxOk ns3::WifiMacHeader (DATA ToDS=0, FromDS=0,
MoreFrag=0, Retry=0, MoreData=0 Duration/ID=0us,
DA=ff:ff:ff:ff:ff:ff, SA=00:00:00:00:00:14,
BSSID=ff:ff:ff:ff:ff:ff, FragNumber=0, SeqNumber=0)
ns3::LlcSnapHeader (type 0x800) ns3::Ipv4Header (tos
0x0 DSCP Default ECN Not-ECT ttl 1 id 0 protocol 17
offset (bytes) 0 flags [none] length: 48 10.1.0.20 >
10.1.0.19) ns3::UdpHeader (length: 28 654 > 654)
ns3::aodv::TypeHeader (RRER) ns3::aodv::RerrHeader
(Unreachable destination( ipv4 address, seq.
number):10.1.0.1, 0No delete flag 0)
ns3::WifiMacTrailer ()
```

Gambar 2.9 *Trace* pengiriman *Route Error* (RERR)

Route Error memiliki ciri utama yaitu ns3::aodv::TypeHeader (RRER). Terdapat informasi IP mana yang hilang. Informasi IP yang hilang bisa dimuat lebih dari satu dengan pemisah "," (koma) dan diakhiri dengan ",0" sebagai akhir daftar ip yang hilang. Pada RERR juga diperlukan ACK setelah menerima, sama seperti RREP.

Gambar 2.10 merupakan contoh CTL_ACK pada NS-3 *Trace File*.

```
t                                                          3.16784
/NodeList/8/DeviceList/0/$ns3::WifiNetDevice/Phy/State/Tx
ns3::WifiMacHeader        (CTL_ACK        Duration/ID=0us,
RA=00:00:00:00:00:01) ns3::WifiMacTrailer ()
```

Gambar 2.10 Contoh CTL_ACK

ACK bisa jadi merupakan baris yang paling sering muncul pada NS-3 Trace File, karena hampir di setiap pengiriman pasti diikuti dengan ACK. Ciri utamanya adalah terdapat kata CTL_ACK.

[Halaman ini sengaja dikosongkan]

BAB III
PERANCANGAN

Perancangan merupakan bagian penting dari pembuatan sistem secara teknis. Sehingga bab ini secara khusus akan menjelaskan perancangan sistem yang dibuat dalam Tugas Akhir ini. Berawal dari deskripsi umum sistem hingga perancangan skenario alur dan implementasinya. Terdapat beberapa istilah yang perlu dimengerti sebelumnya untuk memudahkan pembaca. Istilah-istilah tersebut dicantumkan pada tabel 3.1.

Tabel 3.1 Tabel Istilah pada Tugas Akhir

Istilah	Singkatan	Arti/Keterangan
Network Simulator 3	NS-3	Merupakan aplikasi simulator jaringan versi 3
Total Weight of Route	TWR	Merupakan formula untuk menghitung bobot suatu *node* dengan *node* tetangganya berdasarkan kecepatan, percepatan, dan arah
Dynamic Beacon Scheduling	DBS	Merupakan formula untuk menentukan nilai *Hello Interval* berdasarkan kondisi *node* tetangga yang ditunjukkan oleh nilai TWR.
βmax	-	Merupakan batas atas nilai *hello interval* pada formula DBS
Bmin	-	Merupakan batas bawah nilai *hello interval* pada formula DBS
TWRmax	-	Merupakan batas atas nilai bobot *node* dengan *node*

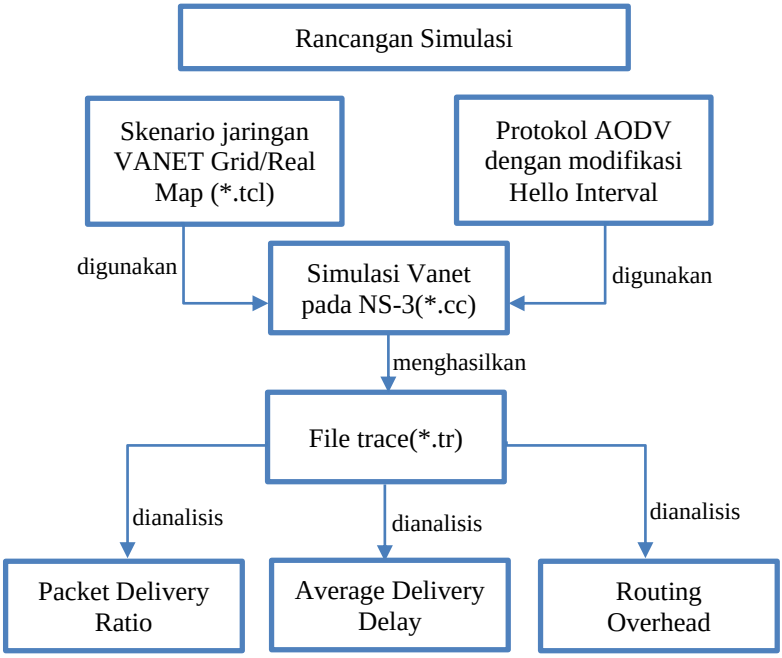
Istilah	Singkatan	Arti/Keterangan
		tetangganya yang akan digunakan pada formula DBS.
TWRmin	-	Merupakan batas bawah nilai bobot *node* dengan *node* tetangganya yang akan digunakan pada formula DBS.
Packet Delivery Ratio	PDR	Merupakan perbandingan antara paket yang dikirim *node source* dengan paket yang diterima *node destination*.
Average Delivery Delay	-	Merupakan rata-rata *delay* antara waktu paket yang diterima dan waktu paket kirim.
Routing Overhead	RO	Merupakan jumlah paket *control routing* yang di transmisikan per data paket yang terkirim ke tujuan selama simulasi terjadi.

3.1 Deskripsi Umum

Pada tugas Akhir ini akan dilakukan analisis pengaruh *Hello Interval* pada protokol dari MANET jenis reaktif yaitu AODV di jaringan VANET. Ilustrasi rancangan simulasi dapat dilihat pada Gambar 3.1.

Dalam pengujian ini, terdapat 2 jenis skenario yang digunakan sebagai perbandingan pengukuran yaitu peta berbentuk grid dan peta riil lingkungan lalu lintas kota Surabaya. Skenario dibuat menggunakan bantuan aplikasi SUMO, peta digital OpenStreetMap, dan aplikasi JOSM. Sedangkan untuk simulasi

menggunakan program pada NS-3. Modifikasi adaptif *Hello Interval* dilakukan pada NS-3 dengan mengubah konfigurasi pada modul dari routing protocol AODV



Gambar 3.1 Diagram rancangan simulasi

3.2 Perancangan *Shared Folder*

Dalam perancangan lintas sistem operasi antara pembuatan skenario yang dirancang di sistem operasi Windows sementara program NS-3 yang dijalankan pada sistem operasi Linux dengan lingkungan VMware Workstation, maka pada tugas akhir ini diperlukan *folder sharing* untuk memudahkan pemindahan ataupun duplikasi *file* yang diperlukan dalam simulasi.

3.3 Perancangan Skenario

Perancangan skenario mobilitas uji coba VANET dimulai dari perancangan peta pergerakan *node*, pembuatan rute lalu lintas dan implementasi pergerakan. Dalam Tugas Akhir ini, peta pergerakan *node* yang akan digunakan ada 2 macam yaitu peta grid dan peta riil wilayah Surabaya. Peta grid yang dimaksud merupakan jalan-jalan yang saling berpotongan dan membentuk petak yang menggambarkan contoh lingkungan peta riil dalam bentuk sederhana. Peta grid digunakan sebagai tes awal implementasi VANET karena peta grid seimbang dan stabil. Sedangkan untuk peta riil adalah peta yang menggambarkan lingkungan lalu lintas yang nyata. Peta riil yang digunakan yaitu lingkungan lalu lintas kota Surabaya. Peta kota Surabaya merupakan hasil impor dari OpenStreetMap. Untuk penjelasan masing-masing peta adalah sebagai berikut:

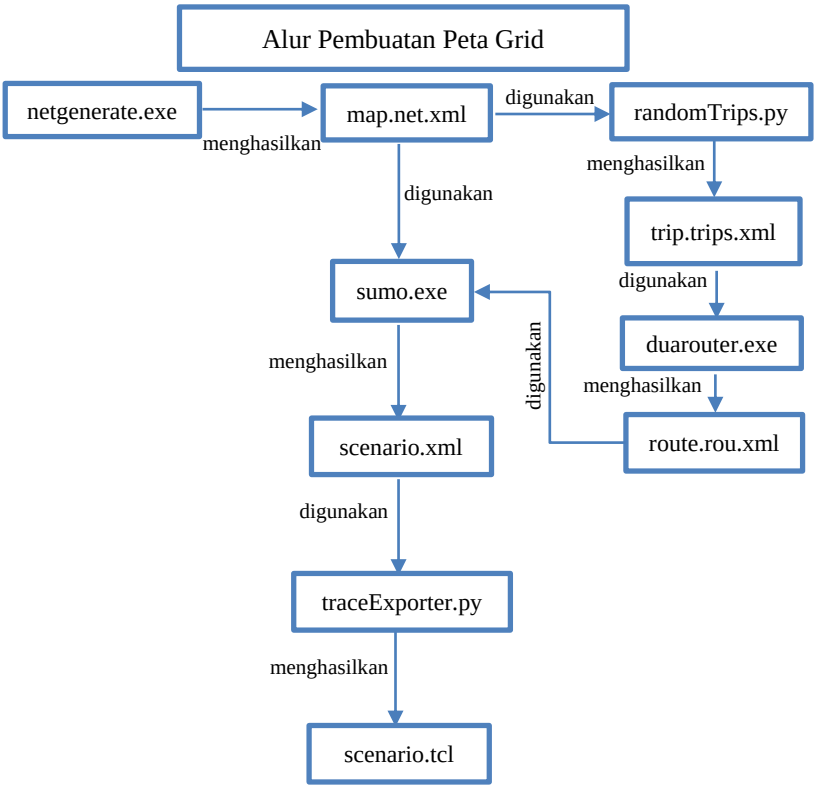
3.3.1 Pembuatan Peta Grid

Pembuatan peta grid diawali dengan menentukan panjang dan lebar peta grid yang digunakan. Setelah ukuran peta ditentukan, langkah berikutnya adalah menentukan banyak garis horizontal dan garis vertikal. Peta yang digunakan berukuran 1500m x 1500m. Jumlah garis horizontal dan vertikalnya masing-masing sebanyak 10 (sepuluh). Ilustrasi alur pembuatan peta grid dapat dilihat pada Gambar 3.2.

Pembuatan peta dengan konfigurasi tersebut menggunakan *tools* netgenerate yang menghasilkan map dalam bentuk grid yang dapat dilengkapi dengan *traffic light* dalam setiap persimpangannya. Pada tugas akhir ini, kecepatan node atau kendaraan yang digunakan adalah 20 m/s. Hasil peta tadi digunakan untuk *node* beserta titik awal dan tujuannya secara acak pada peta dengan modul randomTrips pada SUMO. Secara *default* randomTrips akan mendeploy *node* ke-n pada detik ke n, yang berakibat pada simulasi yang kita lakukan jika waktu simulasi yang

kita lakukan kurang dari n. Atau dengan kata lain terdapat *node* yang tidak bergerak. Oleh karena itu, akan dilakukan sedikit modifikasi pada *file* hasil randomTrips agar semua node di definisi di awal (detik ke-0).

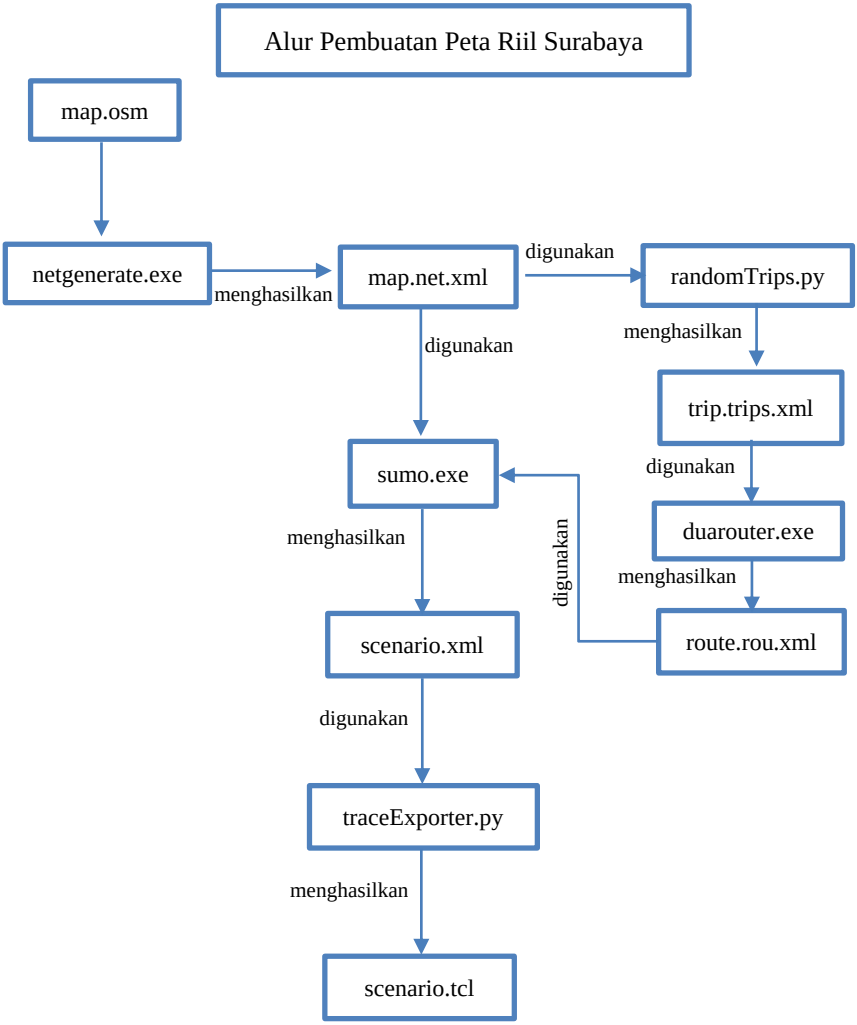
Hasil modifikasi tersebut akan digunakan oleh duarouter untuk memberikan rute masing-masing *node*. Kedua *file* baik peta grid dan pergerakan disimulasikan menjadi satu kesatuan dalam konfigurasi simulasi dan dengan menggunakan *tools* sumo dan modul traceExporter menghasilkan skenario yang dapat digunakan pada NS-3.



Gambar 3.2 Alur pembuatan peta grid

3.3.2 Pembuatan Peta Riil Surabaya

Pembuatan peta riil Surabaya diawali dengan mencari daerah umum di sekitar kota Surabaya yang mendekati bentuk grid dan dengan mobilitas tinggi yang cocok untuk jaringan VANET. Agar



Gambar 3.3 Alur pembuatan skenario riil

pembuatan peta mendekati dengan aslinya, maka OpenStreetMap digunakan untuk membantu melakukan *capture* peta. Kemudian dilakukan modifikasi atau penyempurnaan dengan tools JOSM. Penyempurnaan dapat dilakukan pada jalan yang terputus saat melakukan *capture* atau pengaturan *traffic light*. Setelah dirasa sempurna, peta sudah dapat dikonversi ke dalam bentuk .net.xml menggunakan *tools* netconvert. Tahap selanjutnya sama dengan pembuatan skenario grid hingga menghasilkan jaringan riil VANET dengan skenario kota Surabaya yang bisa berjalan pada NS-3. Ilustrasi alur pembuatan peta riil Surabaya terdapat pada Gambar 3.3.

3.4 Perancangan Modifikasi Protokol dengan TWR

Pada Tugas Akhir ini, modifikasi *Hello Interval* dilakukan dengan mempertibangkan parameter kecepatan, percepatan dan arah terhadap *node* berikutnya (*neighbor node*). *Total Weight of Route* (TWR) [4] merupakan formula yang dapat menggabungkan ketiga parameter tersebut. Nilai parameter kecepatan, percepatan dan posisi node dapat dikirimkan melalui *Hello Message*. Secara default, *Hello Message* tidak mengirimkan informasi mengenai kecepatan, percepatan dan jarak suatu node. Untuk itu diperlukan adanya modifikasi terhadap struktur paket *Hello Message*. Modifikasi *Hello Message* dilakukan dengan menambahkan *field Speed, Acceleration,* dan koordinat x dan y. Perubahan struktur paket *Hello* dapat dilihat pada tabel 3.2.

Tabel 3.2 Modifikasi Struktur Paket *Hello*

Dst IP Address	*Dst Sequence Number*	*Hop Count*
Lifetime	*Speed*	*Acceleration*
x		y

Dalam Tugas Akhir ini, simulasi dilakukan dengan *node source* dan *node destination* berada dalam posisi diam (*static node*). Selain itu, penggunaan TWR pada Tugas Akhir ini hanya

untuk jaringan local (semua node dan node tetangganya), maka perlu dilakukan penyesuaian terhadap formula TWR. Perhitungan TWR terdari atas empat faktor, yaitu kecepatan, percepatan, arah, dan kualitas hubungan.

Faktor kecepatan, percepatan dan arah didapat dari selisih nilai yang dimiliki oleh *next-hop node* terhadap *node* tujuan. Namun pada tugas akhir ini tidak mementingkan arah dari *node* tujuan ataupun *node* sumber, karena hanya mementingkan *node* tetangganya. Hal ini menyebabkan faktor arah menjadi tidak relevan. Agar tidak terjadi pengurangan bobot pada TWR, maka faktor arah diubah menjadi faktor jarak antara *next-hop node* dengan *node* yang bersangkutan. Selain itu faktor kualitas hubungan juga tidak relevan pada tugas akhir ini. Alasannya adalah dalam modifikasi *Hello Interval* yang dipentingkan adalah ketersediaan *neighbor*. Perubahan formula pada TWR ditunjukkan pada Persamaan 3.1.

$$TWR = f_s \times |S_n - S_d| + f_a \times |A_n - A_d| + f_d \times dist(n,d) \quad (3.1)$$

dimana,
dist (n,d) : Jarak *node* dengan *next-hop node*.

Semua nilai parameter yang diperlukan diatas didapatkan dari paket *Hello* yang dikirimkan secara periodik oleh *neighbor node*. Setelah perhitungan TWR didapat, akan dilakukan implementasi adaptif *Hello Interval* berdasarkan nilai TWR yang di dapat. Penentuan nilai *interval* didapat dari nilai *threshold* yang didapat dari uji coba protokol AODV original.

3.5 Perancangan *Dynamic Beacon Scheduling(DBS)*

Secara default *Dynamic Beacon Scheduling* [5] menggunakan kecepatan sebagai acuan penentuan nilai *Hello Interval*. Pada tugas akhir ini akan dilakukan sedikit modifikasi terhadap formula *Dynamic Beacon Scheduling*. Adapun perubahan formula yang terjadi ditunjukkan pada Persamaan 3.2.

$$\beta_i = \begin{cases} \beta_{min} \;\; if\; TWR_i \geq TWR_{max} \\ \frac{TWR_{max}}{TWR_i} * \beta_{min} \;\; if\; TWR_{min} \leq TWR_i \leq TWR_{max} \\ \beta_{max} \;\; if\; TWR_i \leq TWR_{min} \end{cases} \quad (3.2)$$

dimana,
TWR_{max} : TWR maksimum yang terjadi dalam skenario
TWR_{min} : TWR minimum yang terjadi dalam skenario

Penentuan nilai β_{min} dan β_{max} dilakukan dengan melakukan uji coba pada protokol AODV default dengan mengganti parameter *Hello Interval* secara manual. Kemudian dicari nilai *Hello Interval* dibawah 1 yang menghasilkan PDR tertinggi untuk meng-*handle* kendaraan yang riskan kehilangan *neighbor* yang selanjutnya disebut sebagai *βmin*. Selain itu, ditentukan pula nilai *Hello Interval* diatas 1 yang menghasilkan PDR tertinggi untuk meng-*handle* kendaraan yang tidak mudah kehilangan *neighbor*.

Begitu pula dengan nilai TWR_{max} dan TWR_{min}. Ditentukan dengan melakukan uji coba pada protokol AODV default untuk nilai *Hello Interval* yang telah ditentukan pada percobaan sebelumnya.

3.6 Perancangan Simulasi pada NS-3

Pada perancangan kode NS-3 dengan konfigurasi VANET, dilakukan penggabungan skenario mobilitas dengan skrip TCL serta kode program vanet-routing-compare dari NS-3. Berikut perancangan sistem VANET yang ditunjukkan oleh Tabel 3.3.

Tabel 3.3 Parameter – parameter simulasi

No.	Parameter	Spesifikasi
1	Network simulator	NS-3, 3.26
2	Routing Protocol	AODV dengan adaptif Hello Interval
3	Waktu Simulasi	200 detik
4	Area simulasi	1500m x 1500m

No.	Parameter	Spesifikasi
5	Banyak kendaraan	50, 75, 100
6	Radius transmisi	320 [12]
7	Kecepatan maksimal	20 m/s
8	Tipe data	Constant Bit Rate (CBR)
9	Source / Destination	Statik (Node 1 / Node 0)
10	Kecepatan generasi paket	1 paket per detik
11	Ukuran paket data	64 bytes
12	Protocol MAC	IEEE 802.11p
13	Mode propagasi	Two-Way ground reflection model
14	Mobility model	Peta grid dan Peta riil Surabaya
15	Tipe Mobility	NS-2
16	Tipe Kanal	Wireless channel

3.7 Perancangan Metrik Analisis

Metrik yang akan dianalisis pada Tugas Akhir ini adalah sebagai berikut :

3.7.1 *Packet Delivery Ratio* (PDR)

Packet delivery ratio dihitung dari perbandingan antara paket yang dikirim dengan paket yang diterima [13]. *Packet Delivery Ratio* dihitung dengan menggunakan Persamaan 3.3, dimana *received* adalah banyaknya paket yang diterima dan *sent* adalah banyaknya paket yang dikirimkan.

$$PDR = \frac{received}{sent} \times 100\% \tag{3.3}$$

3.7.2 *Average Delivery Delay*

Average delivery delay dihitung dari rata-rata *delay* antara waktu paket yang diterima dan waktu paket kirim [13]. *Average delivery delay* dihitung dengan Persamaan 3.4, dimana $t_{received[i]}$

dalam satuan detik adalah waktu penerimaan paket dengan urutan/id ke-I-, $t_{sent[i]}$ dalam satuan detik pula adalah waktu pengiriman paket dengan urutan/id ke-*I*, dan sent merupakan banyaknya paket data yang dikirimkan. Pada akhirnya nilai Delay menggunakan satuan detik.

$$Delay = \frac{\sum_{i \leq sent}^{i=0} t_{received[i]} - t_{sent[i]}}{sent} \quad (3.4)$$

3.7.3 *Routing Overhead* (RO)

Routing overhead merupakan jumlah paket *control routing* yang di transmisikan per data paket yang terkirim ke tujuan selama simulasi terjadi. *Routing overhead* dihitung berdasarkan paket routing yang ditransmisikan baik itu *Route Request* (RREQ), *Route Reply* (RREP), dan *Route Error* (RERR).

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Bab ini membahas implementasi perancangan perangkat lunak dari aplikasi yang merupakan penerapan 41cenario, kebutuhan simulasi dan alur sistem yang mengacu pada desain dan perancangan yang telah dibahas sebelumnya. Selain itu, bab ini juga membahas lingkungan pembangunan perangkat lunak yang menjelaskan spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam pembangunan sistem.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dibagi menjadi dua bagian, meliputi perangkat lunak dan perangkat keras.

4.1.1 Perangkat Lunak

Lingkungan implementasi dan pengembangan dilakukan menggunakan perangkat lunak sebagai berikut:

- Sistem Operasi Linux Mint 17.3 Rosa 64-bit sebagai lingkungan pengembangan NS-3 dan Sistem Operasi Windows 8 64 bit untuk lingkungan SUMO,
- Vmware Workstation untuk pengembangan aplikasi NS-3 dan analisis,
- SUMO versi 0.29.0 untuk mendesain skenario mobilitas VANET,
- Google Chrome versi 58.0.3029.110 (64-bit) untuk mengoperasikan web OpenStreetMap, dan
- JOSM versi 11526 sebagai editor peta hasil ekspor dari OpenStreeMap.

4.1.2 Perangkat Keras

Lingkungan perangkat keras yang digunakan selama proses pengerjaan Tugas Akhir adalah sebgai berikut:

- *Processor* Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz,
- *Installed Memory* (RAM) 12.00 GB DDR3,
- Media penyimpanan sebesar 1 TB.

4.2 Implementasi *Shared Folder*

Untuk memudahkan *transfer* dan duplikasi data antara dua Sistem Operasi berbeda diperlukan *Shared Folder*. *Shared Folder* dibuat pada sistem operasi Linux (*guest*) sebagai lingkungan pengembangan simulasi dan akan diakses oleh sistem operasi Windows. Implementasi pembuatan *shared folder* antara Windows dan Wmware Workstation dicantumkan secara lengkap pada lampiran 1.

4.3 Implementasi Skenario

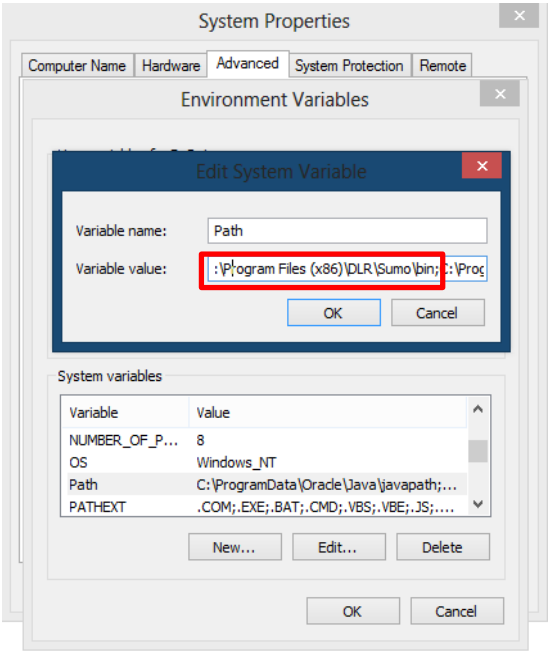
Implementasi skenario mobilitas VANET dibagi menjadi 2 bagian, yaitu implementasi skenario grid dan skenario riil kota Surabaya. Masing – masing jenis skenario dibuat sebanyak 10 buah mobilitas berbeda.

4.3.1 Skenario Grid

Implementasi skenario grid akan dilakukan dengan menggunakan *tools* yang ada pada sumo. Adapun peta yang akan dibuat memiliki luas total 1500 m x 1500 m dengan jumlah petak vertikal maupun horizontal sama yaitu 10 petak. Untuk itu diperlukan 11 titik secara horizontal dan vertikal dengan luasan per petak 150 m x 150 m.

Selain luasan, kecepatan kendaraan pada grid juga diatur. Pada tugas akhir ini kecepatan kendaraan diatur yaitu maksimum sebesar 20 m/s. Agar memudahkan pekerjaan, setelah melakukan instalasi sumo disarankan untuk mengupdate *path* SUMO pada

Environment Variable pada *Control Panel*. Untuk lebih jelasnya dapat dilihat pada gambar 4.1.



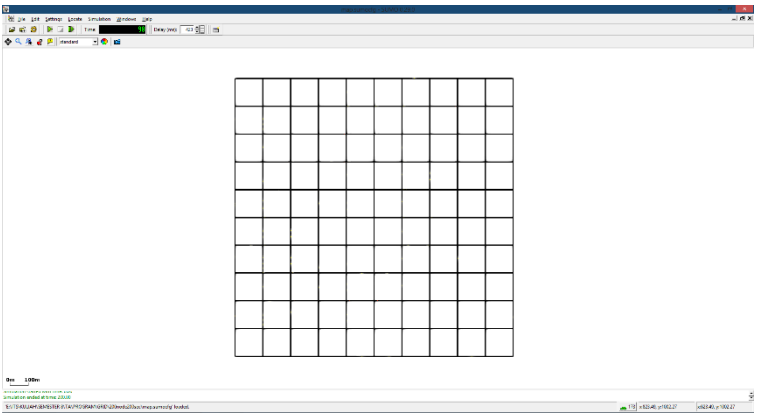
Gambar 4.1 *Update path* SUMO pada *Environtment Variable*

Setelah melakukan *update path* tersebut, pengerjaan skenario dengan menggunakan *tools* SUMO dapat dilakukan di direktori mana saja. Untuk membuat peta, buka *Command Prompt* kemudian lakukan perintah seperti pada gambar 4.2.

```
netgenerate --grid --grid.number=11
--grid.length=150 --default.speed=20 --tls.guess=1
--output-file=map.net.xml
```

Gambar 4.2 Perintah untuk membuat peta

Gambar hasil peta yang dibuat dengan netgenerate dapat dilihat pada gambar 4.3.



Gambar 4.3 Peta hasil netgenerate

Setelah peta terbentuk, maka dilakukan pembuatan *node* beserta posisi asal dan tujuan *node* dengan menggunakan modul randomTrips.py seperti pada gambar 4.4.

```
randomTrips.py -n map.net.xml -e 100 -l
--trip-attributes="departLane=\"best\"
departSpeed=\"max\" departPos=\"random_free\"" -o
trip.trips.xml
```

Gambar 4.4 Perintah untuk membuat *node* beserta asal dan tujuannya

Secara *default* pembuatan *node* pada sumo dilakukan bersamaan dengan detik simulasi. Jadi, *node* ke 100 baru di definisi pada detik ke 100. Hal tersebut dapat menghambat simulasi yang memerlukan 100 *node* namun waktu simulasi yang kurang dari 100 detik. Sehingga diperlukan modifikasi terhadap file trip.trips.xml.
Gambar 4.5 menunjukkan hasil randomTrips.py secara *default*.

```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSc
    <trip id="0" depart="0.00" from="6/6to5/6"
    <trip id="1" depart="1.00" from="4/2to4/1"
```

Gambar 4.5 Hasil randomTrips.py secara default

Definisi *node* ke n yang dilakukan pada detik ke n ditunjukkan oleh kata *depart* pada gambar 4.5. Jadi perlu dimodifikasi nilai dari parameter *depart* menjadi 0.00 agar semua kendaraan di definisikan pada detik ke 0, sehingga semua kendaraan dapat bergerak bersamaan setelah detik ke 0.

Modifikasi dapat dilakukan menggunakan script AWK. Gambar 4.6 menunjukkan *script* AWK yang digunakan pada tugas akhir ini.

```
{
        if ($1=="<trip" && $3!="depart=\"0.00\"") {
                $3="depart=\"0.00\"";
        }
        print;
}
```

Gambar 4.6 *Script* AWK untuk modifikasi trip.trips.xml

Untuk menjalankan *script* tersebut sekaligus mengganti isi dari trip.trips.xml yang sebelumnya, lakukan perintah pada gambar 4.7.

```
awk -f rep.awk trip.trips.xml >> out.xml & move
out.xml trip.trips.xml
```

Gambar 4.7 Menjalankan *script* awk dan memindahkan ke trip.trips.xml

Hasil dari perintah diatas dapat dilihat pada gambar 4.8.

```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSc
    <trip id="0" depart="0.00" from="6/6to5/6"
    <trip id="1" depart="0.00" from="4/2to4/1"
    <trip id="2" depart="0.00" from="0/9to1/9"
    <trip id="3" depart="0.00" from="0/1to0/2"
    <trip id="4" depart="0.00" from="10/10to10/
    <trip id="5" depart="0.00" from="2/6to2/5"
    <trip id="6" depart="0.00" from="4/9to5/9"
```

Gambar 4.8 Hasil modifikasi dengan script awk

Dapat dilihat bahwa nilai parameter *depart* telah diubah semua menjadi 0 yang menandakan bahwa kendaraan akan di definisi pada detik ke 0. Berkas trip.trips.xml kemudian dibuatkan rute berdasarkan peta map.net.xml dengan *tools* duarouter.exe. Secara

default algoritma yang digunakan untuk membuat rute adalah djikstra. Perintah untuk membuat rute dapat dilihat pada gambar 4.9.

```
duarouter.exe -n map.net.xml -t trip.trips.xml -o
route.rou.xml --ignore-errors --repair
```

Gambar 4.9 Perintah untuk membuat rute

Langkah berikutnya adalah pembuatan skenario yang merupakan penggabungan file peta dengan rute pergerakan. *Tools* yang digunakan adalah sumo.exe. Output dari *tools* ini adalah file xml yang merupakan skenario pada peta grid dengan rute yang telah didapat dari generate *file* sebelumnya. Perintah untuk membuat skenario tersebut dapat dilihat pada gambar 4.10.

```
sumo -b 0 -e 200 -n map.net.xml -r route.rou.xml
--fcd-output=scenario.xml
```

Gambar 4.10 Perintah untuk membuat skenario

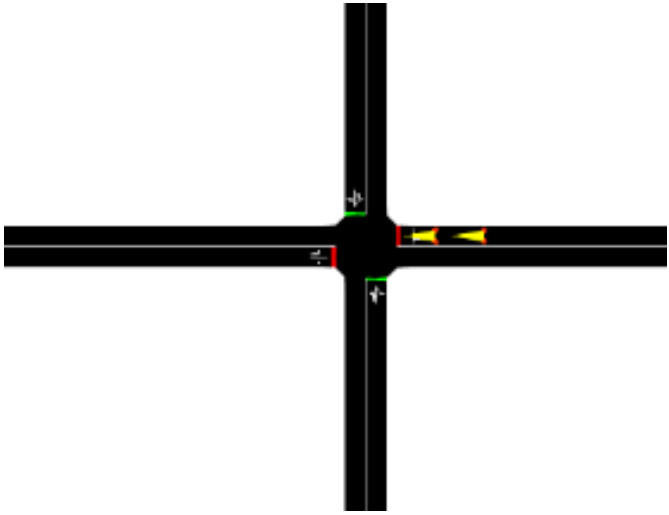
Pada perintah diatas, skenario dimulai pada detik ke 0 ditunjukkan oleh parameter "-b 0" dan diakhiri pada detik ke 200 ditunjukkan oleh parameter "-e 200". Untuk melihat cuplikan pergerakan dari *node* dapat dibuat *file* scenario.sumocfg. *File* sumocfg pada intinya sama seperti perintah sumo.exe, yaitu menggabungkan *file* peta dengan *file* pergerakan. *File* sumocfg dibuat secara manual pada *text editor*. Gambar 4.11 merupakan *file* sumocfg yang dibuat.

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/x
sd/sumoConfiguration.xsd">

      <input>
            <net-file value="map.net.xml"/>
            <route-files value="route1.rou.xml"/>
      </input>
      <time>
            <begin value="0"/>
            <end value="200"/>
```

```
        </time>
</configuration>
```

Gambar 4.11 *File* scenario.sumocfg

File scenario.sumocfg diatas dapat dibuka pada sumo-gui untuk melihat cuplikan pergerakan kendaraan. Cuplikan pergerakan kendaraan dapat dilihat pada gambar 4.12.



Gambar 4.12 Cuplikan pergerakan kendaraan

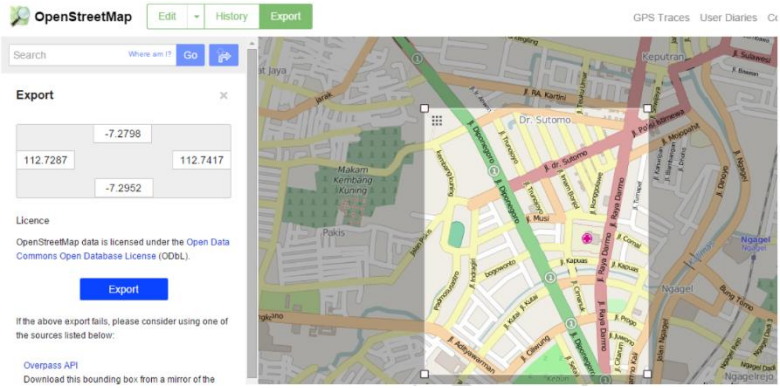
Berkas scenario.xml yang merupakan hasil perintah pada gambar 4.10 kemudian dikonversi menjadi *file* pergerakan dalam format mobilitas NS-2 dengan bantuan modul traceExporter.py. Gambar 4.13 menunjukkan perintah untuk melakukan konversi menjadi format mobilitas NS-2.

```
traceExporter.py --fcd-input=scenario.xml
--ns2mobility-output=scenario.tcl
```

Gambar 4.13 Perintah untuk mengkonversi *file* xml dari SUMO ke dalam format mobilitas NS-2

4.3.2 Skenario Riil

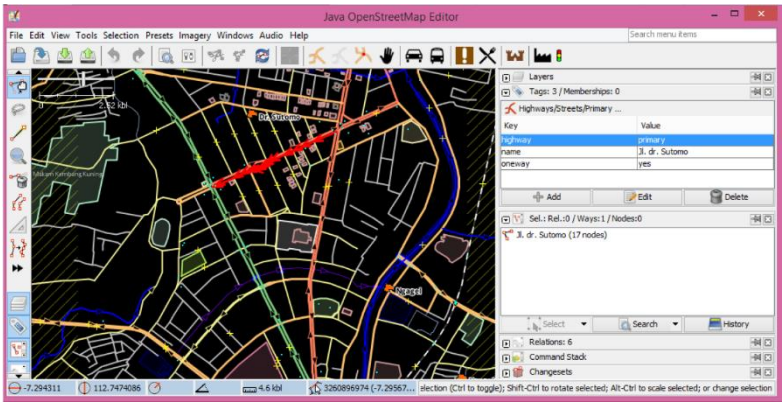
Skenario ini menggunakan peta wilayah Surabaya yang diambil dari OpenStreetMap dengan cara menandai wilayak yang diinginkan kemudian di ekspor atau di download dengan format .osm melalui browser. Untuk lebih jelasnya ditunjukkan pada gambar 4.14.



Gambar 4.14 Proses menandai dan ekspor dari OpenStreetMap

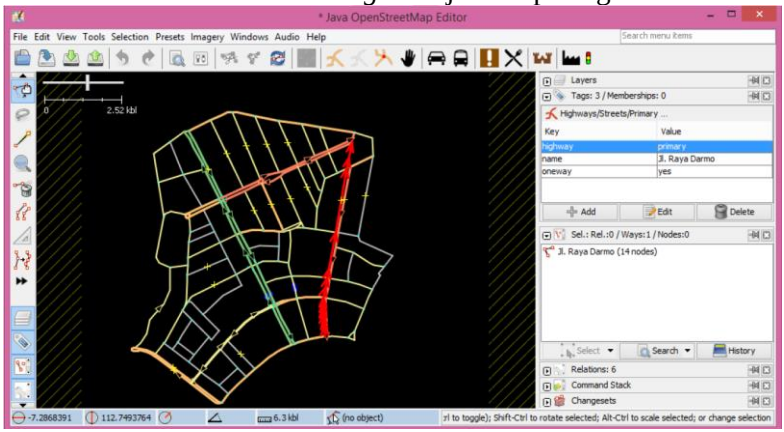
Peta yang sudah diambil dari OpenStreetMap kemudian disunting menggunakan JOSM untuk mengisikan lampu merah pada tiap perempatan. Jalan yang tidak digunakan dihapus serta bentuk dan ukuran peta dibuat supaya hampir sama dengan peta grid. Beberapa jalan baru juga ditambahkan agar kepadatan jalan tetap stabil sehingga tidak akan ada daerah pada peta yang jarang dikunjungi kendaraan. Jalan baru tersebut juga berfungsi agar bentuk peta mendekati persegi agar mirip dengan skenario grid. Kecepatan pada peta nyata Surabaya disesuaikan dengan keadaan nyata juga seperti Jalan Raya Darmo pasti memiliki batas kecepatan yang lebih tinggi daripada Jalan Untung Suropati.

Gambar 4.15 merupakan *screenshot* saat melakukan *editing* pada JOSM.



Gambar 4.15 Proses *editing* peta pada JOSM

Hasil setelah melakukan *editing* ditunjukkan pada gambar 4.6.



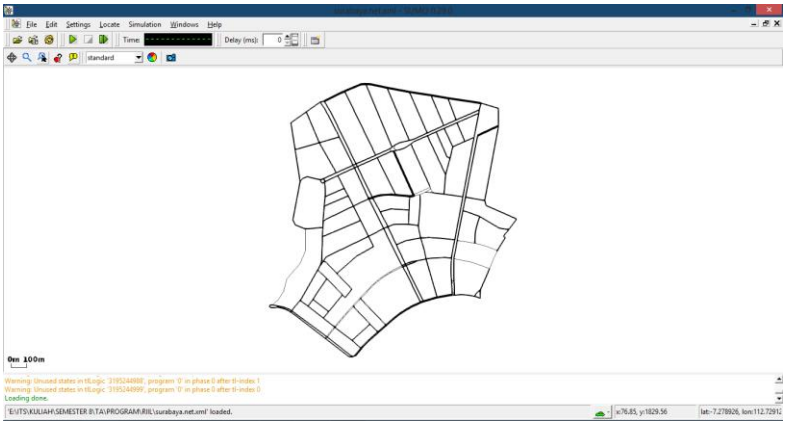
Gambar 4.16 Hasil setelah dilakukan *editing* pada JOSM

Setelah selesai melakukan pengeditan terhadap peta osm, kemudian dilakukan konversi peta menjadi peta net.xml menggunakan *tools* netconvert. Cara konversi dapat dilakukan dengan perintah pada gambar 4.17.

```
netconvert.exe –osm-files map.osm –output-file
map.net.xml
```

Gambar 4.17 Perintah untuk konversi *file* osm menjadi format SUMO

Hasil konversi dari berkas osm dari OpenStreetMap pada SUMO terlihat seperti Gambar 4.18.



Gambar 4.18 Hasil konversi dilihat pada sumo-gui

Setelah peta terbentuk, maka langkah berikutnya sama seperti saat membuat skenario grid sampai berhasil dikonversi kedalam format pergerakan NS-2.

4.4 Modifikasi Protokol dengan TWR

Protokol yang digunakan pada Tugas Akhir ini adalah AODV yang dimodifikasi dengan *Total Weight of Route* (TWR) [4]. TWR memerlukan beberapa parameter yang secara *default* tidak digunakan pada protokol AODV. Adapun parameter-parameter yang digunakan yaitu kecepatan, percepatan dan juga jarak *node* dengan *node* tetangganya. Ketiga parameter tersebut dapat dikirimkan melalui *Hello Message* yang merupakan pesan pertama untuk memberitahu keberadaan node dan mengetahui keadaan *node* tetangganya. Adapun perubahan yang akan dilakukan pada implementasi protokol AODV di NS-3 antara lain sebagai berikut :

- Perubahan struktur paket *Hello*
- Pengiriman paket *Hello*

- Penanganan paket *Hello*
- Kalkulasi TWR

Kode implementasi dari protokol AODV pada NS-3 versi 3.26 berada pada direktori src/aodv/model. Daftar kode sumber yang akan di modifikasi pada direktori tersebut adalah sebagai berikut :

- aodv-packet.h dan aodv-packet.cc untuk mengubah struktur paket *hello* serta perubahan lainnya.
- aodv-rtable.h untuk mendapatkan informasi parameter yang dikirim melalui *hello message*.
- aodv-routing-protocol.h dan aodv-routing-protocol.cc untuk memodifikasi pengiriman paket *hello,* penanganan paket *hello,* dan kalkulasi TWR.

Pada bagian ini, penulis akan menjelaskan langkah-langkah dalam mengimplementasikan protokol dengan modifikasi TWR dengan AODV sebagai dasarnya.

4.4.1 Perubahan Struktur Paket *Hello*

Secara *default Hello Message* tidak mengirimkan parameter-parameter yang dibutuhkan untuk perhitungan TWR. Adapun parameter yang dibutuhkan untuk perhitungan TWR yaitu kecepatan *node,* percepatan *node* dan juga koordinat posisi *node*. Pada dasarnya Hello Message adalah sebuah RREP dengan TTL bernilai 1. Oleh karena itu, struktur paket yang akan dirubah adalah RREP pada aodv-packet.h dan aodv-packet.cc. Adapun atribut yang ditambahkan pada paket RREP adalah m_speed, m_accel, m_x, m_y. Selain itu ada beberapa fungsi yang perlu ditambahkan dan juga di modifikasi. Penambahan atribut ditambahkan pada class RrepHeader seperti ditunjukkan pada gambar 4.19.

```
private:
  uint8_t m_flags;      ///< acknowledgment required flag
  uint8_t m_prefixSize; ///< Prefix Size
  uint8_t m_hopCount;   ///< Hop Count
  Ipv4Address m_dst;    ///< Destination IP Address
  uint32_t m_dstSeqNo;  ///< Destination Sequence Number
  Ipv4Address m_origin; ///< Source IP Address
```

```
uint32_t m_lifeTime;  ///< Lifetime (in milliseconds)

//added field velocity etc
double        m_speed, m_speed1, m_speed2;
double        m_accel, m_accel1, m_accel2;
double        m_x, m_x1, m_x2;
double        m_y, m_y1, m_y2;
```

Gambar 4.19 Penambahan atribut pada paket RREP

Variabel m_speed1, m_speed2 begitu juga yang lainnya merupakan variabel yang membantu meng-*handle* pengiriman paket. Karena nilai dari m_speed berupa *double* sedangkan pengiriman paket mengharuskan pengiriman bertipe *unsigned int* jadi diperlukan konversi. Namun, *Unsigned int* tidak mampu mengkonversi nilai negatif. Untuk itu, diperlukan m_speed1 dan m_speed2 untuk menyimpan nilai didepan koma dan nilai di belakang koma.

Selain penambahan attribut tersebut diperlukan penambahan beberapa fungsi setter dan getter untuk memudahkan penggunaan nilai dari attribut tersebut diatas. Penambahan fungsi *setter* dan *getter* ditunjukkan pada gambar 4.20.

```
void SetSpeed (double s) { m_speed = s; };
double GetSpeed () const { return m_speed; };
void SetAccel (double a) { m_accel = a; };
double GetAccel () const { return m_accel;};
void SetX (double x) { m_x = x;};
double GetX () const { return m_x; };
void SetY (double y) { m_y = y;};
double GetY () const { return m_y; };
```

Gambar 4.20 Penambahan *setter* dan *getter*

Berikutnya yang perlu dimodifikasi pada aodv-packet.h adalah *constructor* pada *class* RrepHeader. Modifikasi yang dilakukan ditunjukkan pada gambar 4.21.

```
RrepHeader (uint8_t prefixSize = 0,
            uint8_t hopCount = 0,
            Ipv4Address dst = Ipv4Address (),
            uint32_t dstSeqNo = 0,
            Ipv4Address origin = Ipv4Address (),
            Time lifetime = MilliSeconds (0),
            /*added field */
```

```
                double speed1 = 0, double speed2 = 0,
                double accel1 = 0, double accel2 =0,
                double x1 = 0, double x2 = 0,
                double y1 = 0, double y2 = 0 );
```

Gambar 4.21 Modifikasi *constructor* pada *class* RrepHeader

Baris dibawah komentar "added field" merupakan attribut yang ditambahkan pada constructor dari class RrepHeader pada *file* aodv-packet.h. *File* aodv-packet.h merupakan header file dari aodv-packet.cc. Oleh karena itu, aodv-packet.cc juga dilakukan modifikasi pada beberapa fungsinya. Yang pertama dimodifikasi adalah *constructor*-nya. Adapun modifikasi yang dilakukan ditunjukkan pada gambar 4.22.

```
RrepHeader::RrepHeader(
      uint8_t prefixSize, uint8_t hopCount,
      Ipv4Address dst, uint32_t dstSeqNo, Ipv4Address
      origin, Time lifeTime,
      /*added field*/
      double speed1, double speed2, double accel1,
      double accel2, double x1, double x2,
      double y1, double y2 ) :
           m_flags (0), m_prefixSize (prefixSize),
           m_hopCount (hopCount), m_dst (dst),
           m_dstSeqNo (dstSeqNo), m_origin (origin),
           /*added field*/
           m_speed1 (speed1), m_speed2 (speed2),
           m_accel1 (accel1), m_accel2 (accel2),
           m_x1 (x1), m_x2 (x2), m_y1 (y1), m_y2 (y2)
{
  m_lifeTime = uint32_t (lifeTime.GetMilliSeconds
()); }
```

Gambar 4.22 Modifikasi *constructor* pada aodv-packet.cc

Berikutnya dilakukan penggantian nilai pada fungsi RrepHeader::GetSerializedSize. Secara *default return value*-nya adalah 19. Karena dilakukan penambahan pengiriman atribut pada paket, ukuran paket pun bertambah. Nilai *return value* pada fungsi ini diganti menjadi 51, karena tambahan atribut yang dikirimkan ada 8 variabel dengan masing-masing variabel membutuhkan 4 bit.

Perubahan terjadi pula pada fungsi penerimaan paket atau RrepHeader::Deserialize. Pada fungsi ini dilakukan konversi nilai

unsigned int menjadi satu nilai *double*. Konversi tersebut ditunjukkan pada gambar 4.23.

```
m_speed1 = (double)i.ReadU32();
m_speed2 = (double)i.ReadU32();
if (m_speed1<1000000 && m_speed2<1000000){
    m_speed = m_speed1+(m_speed2/1000000); }
else if (m_speed1>1000000 && m_speed1<4294967296 &&
m_speed2>1000000 && m_speed2<4294967296)
{
   m_speed = (m_speed1-4294967296) + ((m_speed2-
4294967296)/1000000); }
```

Gambar 4.23 Konversi *unsigned integer* menjadi satu nilai *double*

Variabel m_speed1 digunakan untuk menyimpan nilai di depan koma dan m_speed2 digunakan untuk menyimpan nilai di belakang koma. Ketika pengiriman nilai tersebut telah dikalikan 1000000(1 juta) untuk mendapatkan nilai akurasi yang maksimal. Kondisi pertama digunakan untuk meng-*handle* nilai *speed* yang lebih dari 0 sedangkan kondisi yang kedua digunakan untuk nilai *speed* yang kurang dari 0 (mungkin terjadi untuk variable yang bukan speed). Algoritma diatas berlaku pula untuk percepatan (m_accel) dan juga posisi (m_x, m_y).

4.4.2 Pengiriman Paket *Hello*

Sebelum dilakukan modifikasi pada pengiriman paket *hello*, diperlukan beberapa tambahan atribut. Adapun atribut yang ditambahkan pada aodv-routing-protocol.h ditunjukkan pada gambar 4.24.

```
/// Keep track of the last bcast time
Time m_lastBcastTime;
///added attribute for base speed, acceleration and
direction
double m_speed, m_speed1, m_speed2;
double m_accel, m_accel1, m_accel2;
double m_x, m_x1, m_x2;
double m_y, m_y1, m_y2;
double m_now;
```

Gambar 4.24 Penambahan atribut untuk pengiriman paket *Hello*

Atribut m_now digunakan untuk menyimpan waktu terakhir melakukan *update* atribut lainnya. Hal ini berguna untuk melakukan kalkulasi terhadap percepatan. Berikutnya dilakukan modifikasi pada fungsi SendHello. Pada fungsi ini dilakukan pencarian nilai kecepatan, percepatan serta posisi dari *node* yang kemudian akan dikirimkan sebagai *Hello Message*. Modifikasi pada fungsi SendHello ditunjukkan pada gambar 4.25.

```
//get velocity and position from this node
Ptr<Node> thisNode = socket->GetNode();
Ptr<MobilityModel> thisMobility
       = thisNode->GetObject<MobilityModel>();
Vector thisVelocity = thisMobility->GetVelocity();
Vector thisPosition = thisMobility->GetPosition();
double speed = sqrt((thisVelocity.x*thisVelocity.x)+
                    (thisVelocity.y*thisVelocity.y));

//assign acelleration and update another field
Time now = Simulator::Now();
if(now - m_now != 0)
{
  m_accel = (speed-m_speed)/(now.GetSeconds()-m_now);
  //update position, speed, acelleration
  m_x = thisPosition.x;
  m_y = thisPosition.y;
  m_speed = speed;
  m_now = now.GetSeconds(); //update last update time
}
m_speed2 = modf(m_speed, &m_speed1);
m_speed2 = m_speed2*1000000;
m_accel2 = modf(m_accel, &m_accel1);
m_accel2 = m_accel2*1000000;
m_x2 = modf(m_x, &m_x1);
m_x2 = m_x2*1000000;
m_y2 = modf(m_y, &m_y1);
m_y2 = m_y2*1000000;

RrepHeader helloHeader (/*prefix size=*/ 0,
/*hops=*/ 0,
/*dst=*/ iface.GetLocal (),
/*dst seqno=*/ m_seqNo,
/*origin=*/ iface.GetLocal (),
/*lifetime=*/ Time (m_allowedHelloLoss * m_helloInterval),
/*speed*/ m_speed1, m_speed2,
/*acceleration*/ m_accel1, m_accel2,
/*x*/ m_x1, m_x2, /*y*/ m_y1, m_y2);
```

Gambar 4.25 Modifikasi yang dilakukan pada fungsi SendHello

Pada NS-3 terdapat *smart pointer* yang menampung informasi mengenai kecepatan dan posisi node. Pointer tersebut disebut dengan MobilityModel. Kecepatan, percepatan dan posisi *node* dikirimkan melalui 2 variabel untuk meng-*handle* masalah konversi dari *double* ke *unsigned integer*. Untuk selengkapnya akan dicantumkan pada lampiran buku ini.

4.4.3 Penanganan Paket *Hello*

Pada penanganan paket *hello*, akan dilakukan penyimpanan paket *hello* yang diterima *node* yang kemudian dilakukan pengecekan apakah yang mengirim benar-benar *node* tetangganya. Hello Message yang diterima disimpan dalam sebuah atribut yang merupakan *mapping* dari Ipv4Address dan RoutingTableEntry yang selanjutnya diberi nama m_ipv4AddressEntry. Ipv4Address merupakan alamat IP *node* yang akan digunakan untuk pengecekan *node* tetangga. Sedangkan RoutingTableEntry merupakan sebuah *class* yang menyimpan nilai dari *Hello Message*. Secara *default* RoutingTableEntry tidak menyimpan nilai kecepatan, percepatan dan posisi *node*. Untuk itu diperlukan tambahan atribut dan juga fungsi yang ditunjukkan pada gambar 4.26.

Secara *default* ketika *node* menerima *Hello Message* dari *node* tetangganya, *node* tersebut harus memastikan apakah dia memiliki rute yang aktif kepada *node* tetangga tersebut dan membuat rute jika diperlukan. Proses penyimpanan *Hello Message* pada atribut m_ipv4AddressEntry ditunjukkan pada gambar 4.27.

```
/// Time for which the node is put into the blacklist
Time m_blackListTimeout;

//added field
double m_speed;
double m_accel;
double m_x;
```

```
double m_y;

void SetSpeed (double s) { m_speed = s; };
double GetSpeed () const { return m_speed; };
void SetAccel (double a) { m_accel = a; };
double GetAccel () const { return m_accel;};
void SetX (double x) { m_x = x;};
double GetX () const { return m_x; };
void SetY (double y) { m_y = y;};
double GetY () const { return m_y; };
```

Gambar 4.26 Tambahan attribut pada aodv-rtable.h

```
m_ipv4AddressEntry.insert(std::make_pair
(rrepHeader.GetDst(),newEntry));
```

Gambar 4.27 Penyimpanan *Hello Message* pada m_ipv4AddressEntry

Baris diatas ditambahkan pada fungsi ProcessHello pada *file* aodv-routing-protokol.cc. Selengkapnya akan dicantumkan pada lampiran pada buku ini.

4.4.4 Kalkulasi TWR

Kalkulasi TWR dilakukan dengan cara melakukan perulangan pada atribut *map* yang telah menyimpan paket *Hello Message* yang telah diterima sebelumnya kemudian diambil nilai dari *Hello Message* yang diperlukan untuk kalkulasi TWR. Pengambilan nilai tersebut dilakukan dengan menggunakan fungsi *setter* dan *getter* yang telah ditambahkan sebelumnya. Kalkulasi TWR memerlukan nilai faktor pengali kecepatan, percepatan dan jarak. Maka dari itu perlu didefinisikan terlebih dahulu pada aodv-routing-protocol.h. Pendefinisian faktor pengali tersebut ditunjukkan pada gambar 4.28.

```
double fs = 35, fa = 25, fd = 40;
double TWR_total, nt;
double dx, dy, dis;
double modSpeed;
double modAccel;
double modDistance;
```

Gambar 4.28 Penambahan atribut untuk kalkulasi TWR

Implementasi kalkulasi TWR ditunjukkan pada gambar 4.29.

```
TWR_total=0, nt=0;
for
(std::map<Ipv4Address,RoutingTableEntry>::const_iterator
j = m_ipv4AddressEntry.begin ();
j != m_ipv4AddressEntry.end ();
++j){
    Ipv4Address addr = j->first;
    RoutingTableEntry rte = j->second;

    //check if neighbor or not and calculate TWR
    if (m_nb.IsNeighbor(addr)){
      nt++;
      dx = m_x - rte.GetX();
      dy = m_y - rte.GetY();
      dis = sqrt((dx*dx)+(dy*dy));
      modSpeed = fs * abs(m_speed - rte.GetSpeed());
      modAccel = fa * abs(m_accel - rte.GetAccel());
      modDistance = fd * dis;
      TWR_total = modSpeed + modAccel + modDistance;
  }
}
TWR_total = TWR_total/nt;
```

Gambar 4.29 Implementasi kalkulasi TWR

Definisi TWR_total = 0 pada baris awal digunakan agar nilai TWR tidak bertambah terus menerus. Variabel nt menunjukkan jumlah neighbor saat ini. Jadi nilai TWR yang digunakan untuk kalkulasi interval merupakan rata-rata TWR dari semua *node*. Kode selengkapnya akan dicantumkan pada lampiran buku ini.

4.5 Implementasi *Dynamic Beacon Scheduling* (DBS)

Implementasi *Dynamic Beacon Scheduling* [5] dilakukan berdasarkan persamaan 3.2. Nilai TWR_i merupakan nilai TWR rata-rata yang telah dihitung pada subbab sebelumnya. Penentuan nilai TWR_{max}, TWR_{min}, β_{max}, dan β_{min} akan dijelaskan pada bab selanjutnya. Adapun implementasi perhitungan DBS ditunjukkan pada gambar 4.30.

```
if (TWR_total >= TWRmax) {
    intervalNow = Imin;
} else if (TWRmin<=TWR_total && TWR_total<=TWRmax)
```

```
{
    intervalNow = (TWRmax/TWR_total)*Imin;
} else if (TWR_total <= TWRmin){
    intervalNow = Imax;
}
```

Gambar 4.30 Implementasi DBS

TWR_total merupakan hasil perhitungan rata-rata TWR yang sudah dilakukan sebelumnya. Imin dan Imax merupakan nilai dari β_{max} dan β_{min}. Dan variabel intervalNow merupakan nilai *Hello Interval* yang akan digunakan saat ini. Penggantian nilai *Hello Interval* sekarang dengan nilai intervalNow ditunjukkan pada gambar 4.31.

```
m_helloInterval = Seconds(intervalNow);
```

Gambar 4.31 Penggantian nilai *Hello Interval*

4.6 Implementasi Simulasi pada NS-3

Untuk mensimulasikan VANET pada lingkungan NS-3, dilakukan konfigurasi pada program vanet-routing-compare.cc. Untuk beberapa bagian sistem yaitu pembuatan skenario telah diimplementasikan dengan bantuan SUMO, OpenStreetMap dan JOSM. Sedangkan yang perlu diimplementasikan selanjutnya adalah pembangkitan skenario pada program baik peta grid maupun Surabaya.

Proses pembuatan skenario menggunakan parameter vanet-routing-compare seperti yang ditunjukkan pada Tabel 2.1. Parameter yang dipakai pada penelitian ini antara lain traceFile yang berfungsi menunjuk *file* peta dan pergerakan mana yang akan digunakan, routingTables untuk mencetak ip setiap *node*, TotalSimTime untuk membuat simulasi selesai pada detik ke 200, asciiTrace untuk membuat NS-3 *Trace File* sehingga nantinya bisa dianalisis hasilnya, Sinks dibuat 1 agar *node* pengirim dan penerima ada 1 (node 1 dan 0), dan yang terakhir yaitu txp yang dibuat nilainya 7 agar sesuai dengan penelitian Campolo dan Antonella [13]. Parameter routingProtocol tidak perlu diubah karena *default* protokol yang digunakan pada vanet-routing-

compare adalah AODV. Selain parameter asli dari vanet-routing-compare, ada juga parameter baru yang ditambahkan yaitu trName supaya hasil output NS-3 *Trace File* memiliki nama yang berbeda-beda. Potongan kode selengkapnya akan dicantumkan pada lampiran buku ini.

Gambar 4.32 merupakan potongan kode pemanggilan skenario grid.

```
else if (m_scenario == 3)
    {
        m_traceFile = "scratch/20scene1.tcl";
        m_trName = "real10-1";
        m_routingTables = 1;
        m_nNodes = 50;
        m_TotalSimTime = 200;
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_txp = 7;
    }
```

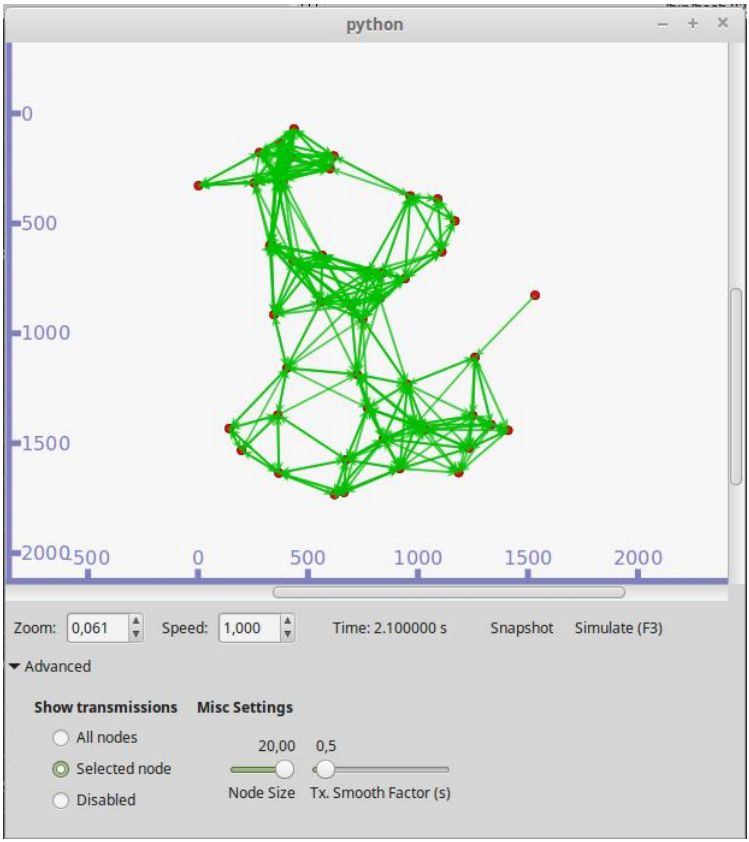
Gambar 4.32 Potongan kode untuk membuat skenario

Ketika menjalankan program disisipkan parameter modul Python bernama PyViz untuk menampilkan animasi perjalanan skenario secara *realtime*. Perintah untuk menjalankan programnya adalah dengan menambahkan --vis setelah menjalankan program pada NS-3. Contoh penggunaannya ditunjukkan pada gambar 4.33.

```
./waf --run "my-vanet-routing-compare --scenario=3"
--vis
```

Gambar 4.33 Contoh perintah untuk menjalankan program dengan tambahan modul PyViz

Pada PyViz ukuran *node* bisa diatur besarnya. Lama kemunculan garis hijau yang menandakan pengiriman paket juga bisa diatur. Kecepatan simulasi juga bisa diatur melalui PyViz. Gambar 4.34 menunjukkan cuplikan animasi saat menjalankan simulasi dengan modul PyViz.



Gambar 4.34 Cuplikan saat menjalankan program NS-3 dengan tambahan modul PyViz

4.7 Implementasi Metrik Analisis

Hasil menjalankan skenario VANET dalam NS-3 dalam bentuk *ASCII Trace* dianalisis dengan tiga metrik yaitu *Packet Delivery Ratio* (PDR), *Delivery Delay*, dan *Routing Overhead*. Untuk *script* lengkapnya akan dicantumkan pada lampiran buku ini.

4.7.1 Implementasi PDR

Gambar 4.35 menunjukkan *pseudocode* untuk melakukan perhitungan PDR.

```
ALGORITMA PDR(trace file)
//Input: trace file simulasi skenario
//Output : jumlah packet sent, packet received, dan
//PDR
BEGIN (
sent <- 0
recv <- 0
pdr <- 0
)
(
If ($1 <- "t" and $3 <-
    "/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Ph
    y/State/Tx" and $9 <- "Retry=0," and $50 ==
    "(size=64)")
        sent +1

If ($1 <- "r" and $3 <-
    "/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/Ph
    y/State/RxOk" and $50 == "(size=64)")
        recv +1
)
END (
pdr <- ( recv / sent ) * 100
print sent
print recv
print pdr
)
```

Gambar 4.35 *Pseudocode* PDR

Paket *sent* didapatkan dengan mencari baris dengan kolom 1 bernilai "t" yang berarti *transmit*. Kolom 3 berisi "/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/State/Tx" yang artinya paket dikirim oleh *node* 1. Kolom 9 berisi "Retry=0" yang artinya hanya pengiriman pada percobaan pertama saja yang akan dihitung. Dan pada kolom 50 berisi "(size=64)" karena data paket berukuran 64 *bytes*. Jumlah baris yang ditemukan disimpan pada variabel *sent*.

Untuk paket *received* didapatkan dengan mencari baris dengan kolom 1 berisi "r" yang berarti *received*. Kolom 3 berisi "/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/Phy/State/Tx" yang artinya paket diterima oleh *node* 0. Kolom 50 berisi "(size=64)" karena data paket berukuran 64 *bytes*. Jumlah baris yang ditemukan untuk paket *received* disimpan pada variabel *recv*.

Segala baris yang ditemukan dihitung jumlahnya untuk kemudian dihitung nantinya lagi untuk mencari presentasi PDR yang didapatkan. PDR dihitung berdasarkan jumlah paket *received* dibagi jumlah paket *sent* dikalikan 100% seperti yang ditunjukkan pada persamaan 3.3.

Contoh perintah untuk analisis PDR pada *trace file* ditunjukkan pada gambar 4.36.

```
awk –f pdr.awk 20scene1.tr
```

Gambar 4.36 Perintah menjalankan *script* pdr.awk

Contoh *output* dari perintah diatas dapat dilihat pada gambar 4.37.

```
Transmitted packet(s) = 540
Received packet(s) = 377
Packet Delivery Ratio = 69.8148%
```

Gambar 4.37 *Output* dari script pdr.awk

4.7.2 Implementasi *Average Delivery Delay*

Mencari *average delivery delay* mirip seperti mencari PDR hanya saja *average delivery delay* akan mencatat waktu yang ada pada kolom 2 dari data *sent* maupun data *received* yang ditemukan. Setelah seluruh baris yang memenuhi didapatkan akan dihitung *delay* yang ada dengan cara mengurangi waktu yang didapat oleh paket *received* dengan waktu paket *sent* dari *id* paket yang sama. Kolom 30 digunakan untuk mencocokkan *id* paket saat mengurangi waktunya. Hasil pengurangan waktu yang didapat dijumlahkan dan dibagi dengan total jumlah paket *received* untuk mendapatkan *average delivery delay*. Gambar 4.38 merupakan *pseudocode* untuk menghitung *Average Delivery Delay*.

```
BEGIN (
for i in pkt_id
```

```
       pkt_id[i] <- 0
for i in pkt_sent
       pkt_sent[i] <- 0
for i in pkt_recv
       pkt_recv[i] <- 0
delay = avg_delay <- 0;
recv <- 0;
)

(
if ( $1 == "t" && $3 ==
     "/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/P
     hy/State/Tx" && $9 == "Retry=0," && $50 ==
     "(size=64)" )
              pkt_sent[$30] <- $2;

if ( $1 == "r" && $3 ==
     "/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/P
     hy/State/RxOk" && $50 == "(size=64)" )
              recv + 1
              pkt_recv[$30] <- $2;
)

END (
for i in pkt_recv
       delay += pkt_recv[i] - pkt_sent[i]
avg_delay = delay / num;
)
```

Gambar 4.38 *Pseudocode* dari *Average Delivery Delay*

Contoh perintah untuk analisis *delay* dari *trace file* dapat dilihat pada gambar 4.39.

```
awk –f average-delivery-delay.awk 20scene1.tr
```

Gambar 4.39 Perintah menjalankan *script* average-delivery-delay.awk

Contoh *output* dari perintah diatas dapat dilihat pada gambar 4.40.

```
Average Packet Delivery Delay = 0.0187855 seconds
```

Gambar 4.40 *Output* hasil *script* average-delivery-delay.awk

4.7.3 Implementasi RO

Jumlah total RO dihitung dengan menjumlahkan RREQ, RREP, dan RRER yang ditemukan. RREQ ditemukan dengan mencari baris yang memiliki kolom 1 berisi “t” yang berarti

transmit. Kolom 50 berisi "(RREQ)" yang berarti tipe paketnya adalah RREQ. Kolom 57 berisi "10.1.0.1" yang berarti *source ip* adalah 10.1.0.1. Dan kolom 63 yang berisi "10.1.0.2" yang berarti *destination ip* adalah 10.1.0.2.

RREP ditemukan dengan mencari baris yang memiliki kolom 1 berisi "t" yang berarti *transmit*. Kolom 50 berisi "(RREP)" yang berarti tipe paketnya adalah RREP. Kolom 60 berisi "10.1.0.1" yang berarti *source ip* adalah 10.1.0.1. Dan kolom 54 yang berisi "10.1.0.2" yang berarti *destination ip* adalah 10.1.0.2.

RRER ditemukan dengan mencari baris yang memiliki kolom 1 berisi "t" yang berarti *transmit*. Kolom 50 berisi "(RRER)" yang berarti tipe paketnya adalah RRER. Kolom 58 berisi "10.1.0.1" yang berarti *source ip* yang tidak ditemukan adalah 10.1.0.1.

Setelah semua didapatkan lalu dihitung total jumlahnya. Gambar 4.41 merupakan *pseudocode* untuk menghitung RO. Contoh perintah untuk menganalisis RO dari *trace file* ditunjukkan pada Gambar 4.42.

Hasil *output* dari analisis RO pada *trace file* ditunjukkan pada Gambar 4.43.

```
BEGIN (
rreq <- 0
rrep <- 0
rerr <- 0
total <- 0
)
(
if($1=="t" && $50 == "(RREQ)" && $57 == "10.1.0.1" && $63 ==
    "10.1.0.2")
                rreq + 1
if($1=="t" && $50 == "(RREP)" && $60 == "10.1.0.1" && $54 ==
"10.1.0.2")
                rrep + 1
if($1=="t" && $50=="(RRER)" && $58 == "10.1.0.1)" )
        rerr + 1
)
END (
total <- rreq + rrep + rerr
print rreq
print rrep
print rerr
print total )
```

Gambar 4.41 *Pseudocode* dari RO

```
awk –f ro.awk 20scene1.tr
```

Gambar 4.42 Perintah menjalankan *script* ro.awk

```
Jumlah RREQ = 864
Jumlah RREP = 714
Jumlah RERR = 30
Jumlah Routing Overhead = 1608
```

Gambar 4.43 *Output* dari *script* ro.awk

BAB V
PENGUJIAN DAN EVALUASI

Pada bab ini akan membahas uji coba dan evaluasi dari sistem yang dibuat.

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada laptop yang telah terpasang perangkat lunak VMware Workstation sehingga terdapat Linux dengan NS3 terpasang di dalamnya. Pada VMware Workstation dibuat sebuah *shared folder* untuk memudahkan proses memindahkan atau duplikasi *file* yang ada pada sistem operasi Windows pada laptop dengan sistem Operasi Linux pada VMware Workstation. Spesifikasi dari laptop yang digunakan untuk uji coba dapat dilihat pada tabel 5.1.

Tabel 5.1 Spesifikasi laptop yang digunakan

Komponen	Spesifikasi
CPU	Processor Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz
Sistem Operasi	Windows 8.0 64-bit
Memori	12 GB 1600 MHz • Default : 4 GB PC3-12800 DDR3 • Added : 8 GB PC3-12800 DDR3
Grafis	Nvidia Geforce GT650M
Penyimpanan	1TB HDD

Untuk konfigurasi dari VMware Workstation yang digunakan untuk uji coba dapat dilihat pada Tabel 5.2.

Tabel 5.2 Spesifikasi VMware Workstation

Komponen	Konfigurasi
Name	Mint 17.3 (Rosa)
Operating System	Other Linux (64bit)
Base Memory	4096 MB
Processor	8

Komponen	Konfigurasi
Storage	50 GB .vmdk file
Network	Bridged

Sebelum dilakukan pengujian terhadap program utama yang digunakan pada Tugas Akhir ini, dilakukan beberapa uji coba sebelumnya untuk menentukan beberapa nilai parameter. Nilai parameter tersebut nantinya akan digunakan pada program utama yang dibuat untuk Tugas Akhir ini. Hasil dari pengujian program utama adalah sebuah *file* berekstensi .tr yang kemudian akan dianalisis dengan *script* AWK yang sudah dibuat sebelumnya.

5.2 Pra Uji Coba

Pada bagian ini akan dijelaskan tentang beberapa uji coba yang perlu dilakukan untuk menentukan nilai-nilai parameter yang diperlukan. Hasil pada uji coba disini akan digunakan pada program utama Tugas akhir ini. Berdasarkan persamaan 3.2, adapun nilai parameter yang diperlukan yaitu βmax, βmin, TWRmax dan juga TWRmin.

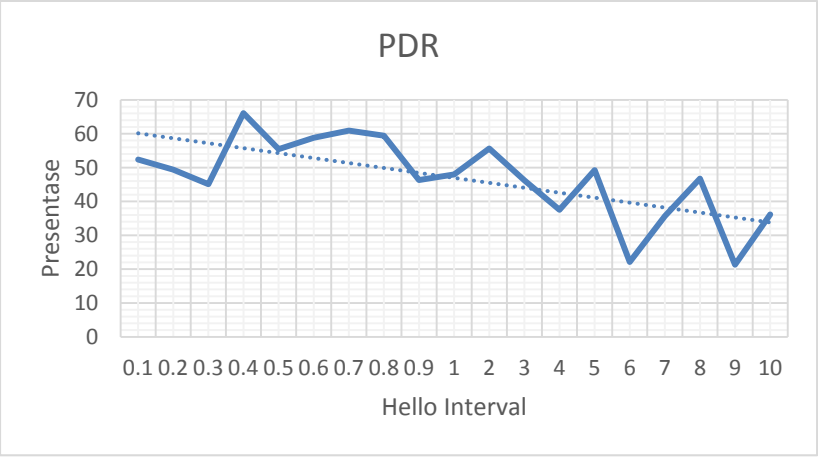
5.2.1 Penentuan nilai βmax dan βmin

Pada Tugas Akhir ini penentuan nilai βmax dan βmin dilakukan dengan cara menjalankan skenario *default* terhadap protokol AODV. Namun terdapat nilai yang divariasi yaitu *Hello Interval*. Secara default nilai *Hello Interval* protokol AODV pada NS-3 adalah 1 detik. Secara teori, semakin sering *hello interval* dikirimkan semakin cepat sebuah *node* melakukan pengenalan terhadap sekitarnya yang mengakibatkan ketersediaan *next-hop node* ketika hendak mengirimkan data paket. Namun hal ini juga berakibat pada jaringan yang terbebani oleh terlalu banyak *Hello Message*, sehingga menghambat pengiriman data paket. Jika dilakukan sebaliknya yaitu menaikan nilai *Hello Interval*, terdapat kemungkinan ketika *next-hop node* dibutuhkan untuk mengirimkan paket ternyata *node* tersebut telah hilang namun

statusnya masih merupakan *neighbor node*. Hal tersebut mengakibatkan kegagalan dalam pengiriman paket.

Jadi skenario yang dilakukan untuk pengujian ini yaitu menggunakan nilai *Hello Interval* 0.1 detik hingga 10 detik. Nilai Hello Interval dibawah 1 detik memiliki variasi perbedaan waktu yaitu 0.1 detik. Jadi nilai *Hello Interval* yang digunakan dibawah 1 detik adalah 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, dan 0.9. Sedangkan untuk nilai Interval diatas 1 detik memiliki variasi perbedaan waktu yaitu 1 detik. Jadi nilai *Hello Interval* yang digunakan diatas 1 detik adalah 1, 2, 3, 4, 5, 6, 7, 8, 9, dan 10.

Gambar 5.1 menunjukkan hasil uji coba yang digunakan pada Tugas Akhir ini.



Gambar 5.1 Grafik hasil uji coba penentuan βmax dan βmin

Skenario hasil uji coba diatas dilakukan dalam 200 detik dengan jumlah kendaraan 50 dan kecepatan maksimum 20 m/s pada peta grid. Letak *node* pengirim dan penerima berada pada ujung peta dalam keadaan diam sedangkan *node* yang lain bergerak. Skenario ini dipilih karena semakin lama waktu simulasi semakin memungkinkan sebuah node pengirim atau penerima tidak memiliki *node* tetangga. Agar mencapai performa yang baik maka ditentukan nilai Intervalnya yaitu antara 0.4 detik dan 5 detik.

Karena PDR ketika *Hello Interval* 0.4 detik memiliki performa tertinggi, sedangkan *Hello Interval* 5 detik dipilih karena nilai PDR masih berada diatas 40%. Nilai 0.4 dan 5 selanjutnya akan disebut nilai βmax dan βmin.

5.2.2 Penentuan nilai TWRmax dan TWRmin

Pada Tugas Akhir ini nilai TWRmax dan TWRmin ditentukan dengan menggunakan skenario yang sama dengan penentuan βmax dan βmin. Skenario disini merupakan hasil generate secara *random* sebanyak 10 kali oleh SUMO dengan spesifikasi yaitu kecepatan maksimum 20 m/s, jumlah *node* 50, *node source* dan *destination* berada pada posisi diam dan waktu simulasi 200 detik. *Hello Interval* yang digunakan yaitu 0.4, 1, dan 5 detik sesuai dengan uji coba sebelumnya. *Hello Interval* 1 detik dimasukkan pula karena merupakan *default* dari protokol AODV pada NS-3. Pada setiap skenario, dilakukan penyimpanan nilai TWR yang kemudian di rata-rata untuk mendapatkan nilai TWR yang mewakili suatu skenario. Berikut hasil uji coba skenario ditunjukkan pada tabel 5.3.

Tabel 5.3 Hasil uji skenario untuk nilai TWR

Scene	**Hello Interval**		
	0.4	**1**	**5**
1	1773.50	1739.32	1837.36
2	1712.93	1715.78	1896.75
3	2043.49	2088.47	2508.97
4	1835.99	1928.75	2113.28
5	2019.47	2067.86	2166.58
6	1837.00	1851.55	2098.23
7	1763.38	1824.55	2092.86
8	1683.49	1821.94	2074.06
9	1908.83	1928.26	1901.18
10	1868.19	1891.36	2051.89
Max	2043.49	2088.47	2508.97
Min	1683.49	1715.78	1837.36

Nilai TWRmax dan TWRmin pada akhirnya ditentukan oleh nilai maximum dan minum dengan *Hello Interval* bernilai 1. Karena nilai *hello interval* ini merupakan *default* untuk AODV pada NS-3. Untuk pertama nilai TWRmax dan TWRmin yang ditentukan yaitu 2000 dan 1700. Berdasarkan hasil tersebut dilakukan testing akhir dengan adaptif *hello interval*. Namun hasil yang didapat menurun dari nilai semula. Oleh karena itu dilakukan variasi terhadap nilai TWRmax dengan nilai yang tidak melebihi hasil TWRmax pada skenario dengan *hello interval* 5 detik (2508.974746). Hasilnya akan disajikan pada subbab berikutnya.

5.3 Hasil Uji Coba

Uji coba yang sebenarnya merupakan penerapan *Hello Interval* yang adaptif pada skenario yang sama dengan pra uji coba diatas. Penentuan nilai *Hello Interval* dilakukan berdasarkan formula *Dynamic Beacon Scheduling* (DBS) [4] atau persamaan 3.2. Nilai βmax yang digunakan adalah 5 detik dan βmin yang digunakan adalah 0.4 detik. Nilai variasi TWRmax yang digunakan adalah 2000, 2100, 2200, dan 2300. Sedangkan nilai TWRmin tidak dilakukan variasi agar memperkecil kemungkinan nilai *Hello Interval* masuk ke interval maksimum. Hasil uji coba pada skenario grid dan Surabaya dapat dilihat sebagai berikut :

5.3.1 Hasil Uji Coba Skenario Grid

Skenario yang di implementasi pada awalnya sama dengan skenario pada penentuan nilai-nilai parameter DBS yaitu 50 kendaraan selama 200 detik dengan maksimum kecepatan 20 m/s dan posisi *node* pengirim dan penerima tetap (tidak bergerak). Tabel 5.4 menyajikan nilai PDR yang dihasilkan dengan variasi nilai TWRmax pada skenario grid dalam persen(%).

Tabel 5.4 Hasil nilai PDR dengan variasi nilai TWRmax

Scene	TWRmax			
	2000	**2100**	**2200**	**2300**
1	0	16.41	22.01	2.40
2	28.50	20.85	97.18	28.59
3	48.35	62.48	66.76	83.35

Scene	TWRmax			
	2000	**2100**	**2200**	**2300**
4	26.03	68.56	53.71	78.58
5	26.92	0	69.81	59.50
6	21.44	1.52	23.27	3.27
7	0	49.13	63.73	46.61
8	13.69	66.22	30.68	52.20
9	28.62	36.39	83.95	39.03
10	88.72	88.60	84.30	81.03
Rata-rata	**28.23**	**41.01**	**59.54**	**47.46**

Dari data diatas dapat disimpulkan bahwa nilai TWRmax yang efektif adalah 2200. Berikutnya akan dilakukan pengujian dengan melakukan variasi terhadap jumlah *node*. Variasi jumlah node yang dilakukan yaitu 75 *node* dan 100 *node*. Hasil pengujian untuk PDR dan untuk metrik lain semua skenario disajikan pada tabel 5.5, tabel 5.6, dan tabel 5.7.

Tabel 5.5 Hasil PDR pada skenario Grid

Jumlah Node	AODV Default	AODV Interval Adaptif
50 Node	47.97%	59.54%
75 Node	56.84%	20.60%
100 Node	33.32%	29.34%

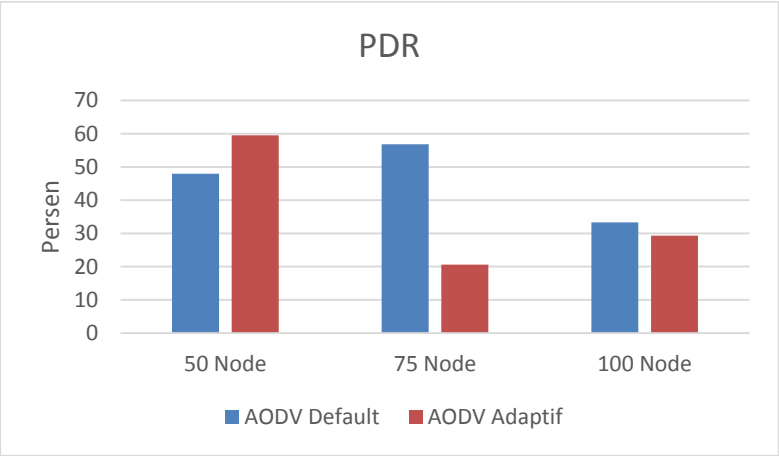
Tabel 5.6 Hasil *Average Delivery Delay* pada skenario Grid

Jumlah Node	AODV Default	AODV Interval Adaptif
50 Node	0.017301065	0.01709214
75 Node	0.067660305	0.13821344
100 Node	0.129071008	0.14476069

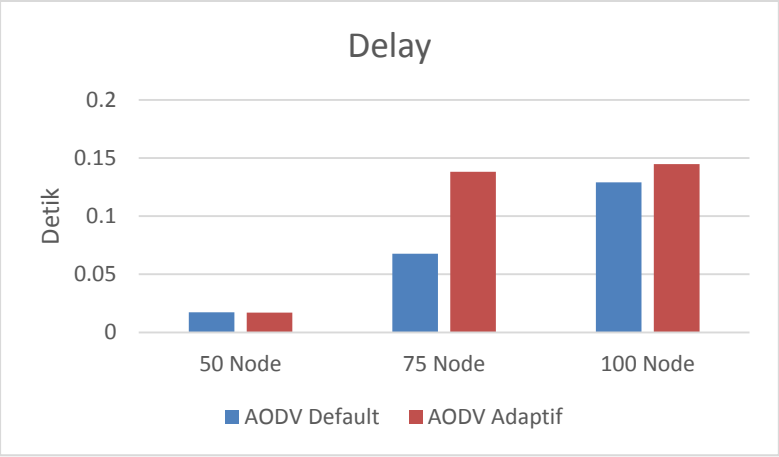
Tabel 5.7 Hasil *Routing Overhead* pada skenario Grid

Jumlah Node	AODV Default	AODV Interval Adaptif
50 Node	1321.8	1896.8
75 Node	3465.4	2089.7
100 Node	5449.1	2286.6

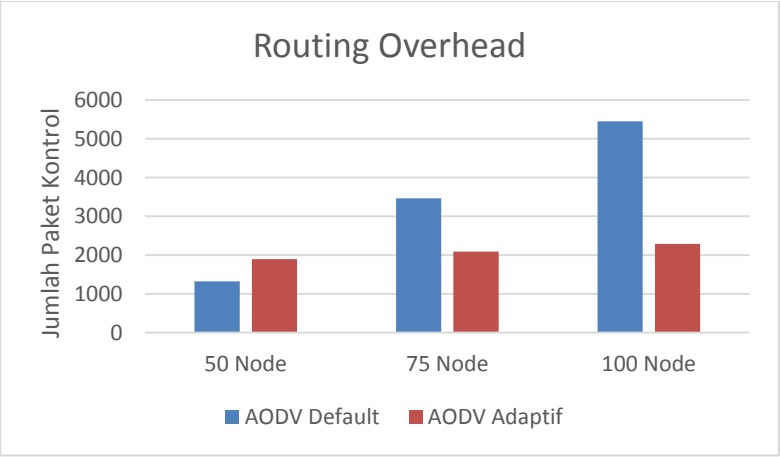
Data diatas disajikan dalam grafik ditunjukkan pada gambar 5.2, 5.3 dan 5.4.



Gambar 5.2 Grafik PDR skenario Grid



Gambar 5.3 Grafik *Delay* skenario Grid



Gambar 5.4 Grafik *Routing Overhead* skenario Grid

Berdasarkan hasil skenario grid dapat dilihat bahwa semakin banyak jumlah *node* dengan *hello interval* yang adaptif membuat nilai PDR semakin menurun. Ini disebabkan oleh nilai TWR [4] yang dihasilkan pada skenario dengan jumlah *node* 75 dan 100 cenderung lebih kecil dari batas bawah (TWRmin) pada rumus *Dynamic Beacon Scheduling* [5]. Dengan kecilnya nilai TWR yang dihasilkan menyebabkan nilai *Hello Interval* cenderung ke arah βmax (5 detik). Selain itu hal ini disebabkan oleh adanya perbedaan waktu pengiriman *hello message*. *Dynamic Beacon Scheduling* [5] membuat *timing* pengiriman *hello message* antara satu *node* dengan *node* lainnya tidak sinkron, sehingga berdampak pada besarnya total RO dan bisa berdampak pada turunnya nilai PDR.

5.3.2 Hasil Uji Coba Skenario Real Wilayah Surabaya

Uji coba riil Surabaya tidak memiliki batas kecepatan yang sama untuk setiap jalannya. Tidak seperti uji coba grid yang memiliki batas kecepatan 20 m/s. Untuk uji coba skenario *real*, kecepatan yang digunakan adalah kecepatan *real* yang diberikan untuk setiap jalannya berdasarkan data pada peta dari

OpenStreetMap. Jalan raya besar akan memiliki batas kecepatan yang tinggi bahkan mencapai 27 m/s, sedangkan jalan kecil seperti jalan perumahan kecepatannya cenderung kecil bahkan ada yang sampai 7 m/s. Pengujian skenario ini dilakukan dengan nilai parameter yang sama dengan skenario grid yaitu 50 kendaran, 75 kendaraan, dan 100 kendaraan dengan waktu simulasi 200 detik dan TWRmax 2200.

Hasil pengujian skenario real peta wilayah Surabaya disajikan pada tabel 5.8, tabel 5.9 dan tabel 5.10.

Tabel 5.8 Hasil PDR pada skenario Real Surabaya

Jumlah Node	AODV Default	AODV Interval Adaptif
50 Node	48.09%	26.30%
75 Node	52.04%	37.54%
100 Node	46.28%	22.39%

Tabel 5.9 Hasil *Average Delivery Delay* pada skenario Real Surabaya

Jumlah Node	AODV Default	AODV Interval Adaptif
50 Node	0.028232	0.0708498
75 Node	0.03513663	0.144458579
100 Node	0.03756988	0.0830854

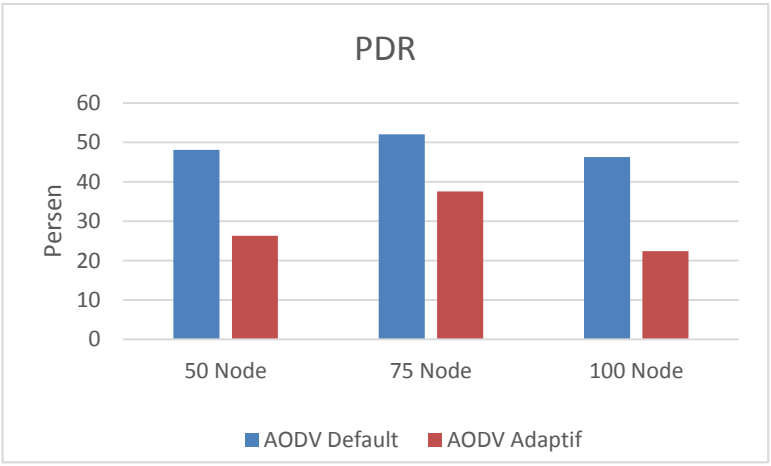
Tabel 5.10 Hasil *Routing Overhead* pada skenario Real Surabaya

Jumlah Node	AODV Default	AODV Interval Adaptif
50 Node	1373.1	1539.9
75 Node	5788.1	2660.6
100 Node	9789.1	2617.1

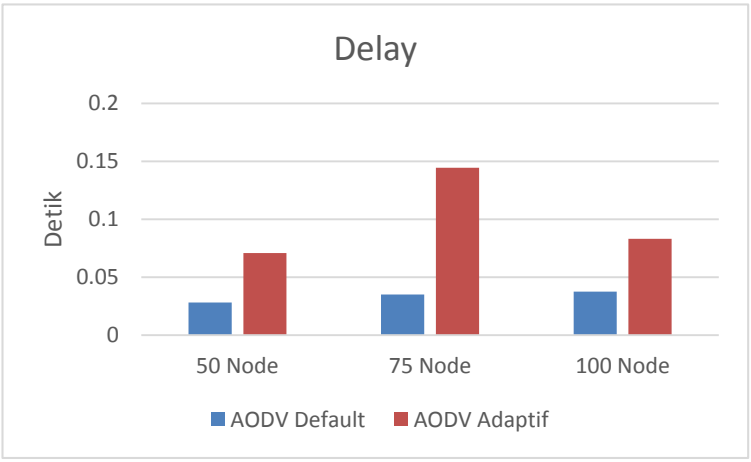
Data diatas disajikan dalam grafik ditunjukkan pada gambar 5.5, 5.6 dan 5.7.

Pada uji coba skenario *real* Surabaya, dapat dilihat bahwa semakin banyak jumlah *node* yang diterapkan dengan nilai *hello interval* yang adaptif menyebabkan penurunan nilai PDR. Kecuali ketika jumlah node berjumlah 75. Hal ini disebabkan oleh penentuan rute pada skenario dengan *node* berjumlah 75 lebih baik

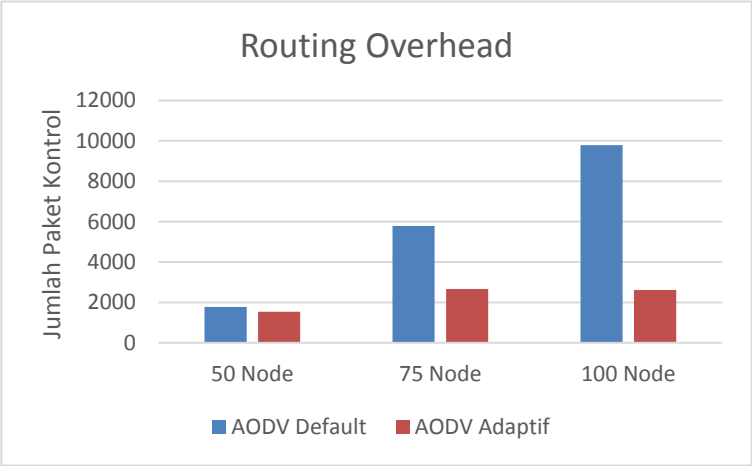
dari pada berjumlah 100. Pada skenario real dengan jumlah *node* 100 terdapat kondisi dimana *node* tidak memiliki *neighbor* sehingga tidak dapat mengirimkan paket.



Gambar 5.5 Grafik PDR skenario real Surabaya



Gambar 5.6 Grafik *Delay* skenario real Surabaya



Gambar 5.7 Grafik *Routing Overhead* skenario real Surabaya

[Halaman ini sengaja dikosongkan]

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir ini serta saran-saran tentang penembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1 Kesimpulan

Dalam proses pengerjaan Tugas Akhir yang melalui tahap perancangan, implementasi, serta uji coba, didapatkan kesimpulan sebagai berikut :

1. Penentuan rentang nilai *Hello Interval* yang adaptif dilakukan dengan cara melakukan uji coba.
2. Pada skenario grid nilai *Hello Interval* yang adaptif memberikan peningkatan PDR dari 47.97% menjadi 59.54% pada skenario dengan jumlah node 50. Namun, mengalami penurunan pada jumlah node 75 yaitu dari 56.84% menjadi 20.60% dan pada jumlah node 100 yaitu dari 33.32% menjadi 29.34%. Sama halnya pada Average Delivery Delay dan juga Routing Overhead. Hal ini disebabkan oleh semakin banyak jumlah node, semakin kecil jarak yang dihasilkan antar node yang mempengaruhi nilai TWR. Selain itu, timing pengiriman hello message menjadi tidak sinkron sehingga berdampak pada jumlah paket hello yang dihasilkan.
3. Pada skenario real Kota Surabaya nilai *Hello Interval* yang adaptif memberikan penurunan PDR dari 48.09% menjadi 26.30% pada skenario dengan jumlah node 50. Pada jumlah node 75 dan 100, PDR juga mengalami penurunan dari 52.04% menjadi 37.54% dan dari 46.28% menjadi 22.39%. Penyebabnya sama seperti yang dijelaskan pada kesimpulan nomor 2.

4. Skenario nyata Surabaya memberikan hasil yang lebih buruk dari skenario grid, ini dikarenakan pada skenario Surabaya kecepatan node lebih beragam dan cenderung lebih tinggi.

6.2 Saran

Saran yang dapat diberikan dalam pengujian sistem ini adalah :

1. Diperlukan penelitian lebih lanjut agar *Hello Interval* menjadi adaptif yang bisa menyesuaikan waktunya berdasarkan keadaan sekitar. Idenya bisa dengan menerapkan waktu *Hello Interval* dengan memainkan jarak transmisi suatu *node*. Awalnya sebuah kendaraan di set jarak transmisinya 320. Namun di jarak ini ternyata node tetangga yang ditemukan kurang jadi jaraknya bisa ditambah sembari mengganti nilai *Hello Interval* berdasarkan jumlah *node*-nya yang dirasa cukup.
2. Peta Surabaya yang dimuat pada OpenStreetMap masih memiliki kesalahan, terutama bagian lampu lalu lintas sehingga agak sulit melakukan implementasi skenario nyata Surabaya menggunakan SUMO. Oleh karena itu masih perlu perbaikan pada OpenStreetMap khususnya daerah Indonesia.
3. Diperlukan lebih banyak lagi penelitian menggunakan NS3 agar target NS3 bisa tercapai yaitu menggantikan NS2 sepenuhnya.

DAFTAR PUSTAKA

[1] N. d. A. L. Khan, "Anycast based Routing in Vanets Using Mobisim," *IDOSI,* 2009.

[2] A. S. a. S. Preetsingh, "Node Selection Algorithm for Routing Protocols in VANET," *International Journal of Advanced Science and Technology,* vol. 96, pp. 43-54, 2016.

[3] G. Kaciak, "Perbaikan Perangkat Komputer," *Padang: Bung Hatta University Press,* 2012.

[4] X. Shen, Y. Wu, Z. Xu dan X. Lin, "AODV-PNT : An Improved Version of AODV Routing Protocol with Predicting Node Trend in VANET," pp. 91-97, November 2014.

[5] M. M. A. a. H. T. Mouftah, "Adaptive Expiration Time for Dynamic Beacon Scheduling in Vehicular Ad-hoc Networks," *IEEE,* 2011.

[6] SUMO, "SUMO Wiki," SUMO, [Online]. Available: http://sumo.dlr.de/wiki/Simulation_of_Urban_MObility_-_Wiki. [Diakses 22 May 2017].

[7] vmware, "Workstation," WMware, [Online]. Available: http://www.vmware.com/products/workstation.html 23/5/2017. [Diakses 23 May 2017].

[8] OpenStreetMap, "About," OpenStreetMap Foundation, [Online]. Available: https://blog.openstreetmap.org/about/. [Diakses 23 May 2017].

[9] OpenStreetMap, "JOSM," OpenStreetMap, [Online]. Available: wiki.openstreetmap.org/wiki/JOSM 23/5/2017.

[Diakses 23 May 2017].

[10] R. Soelistijorini, "Pemrograman Filter," PENS, Surabaya, 2011.

[11] D. I. a. R. Roestam, "Simulasi Model Jaringan Mobile Ad-Hoc (MANET) dengan NS-3," *KNS&I11-052*, 12 November 2011.

[12] R. D. a. P. D. Pankaj Rohal, "Study and Analysis of Throughput, Delay and Packet Delivery Ratio in MANET for Topology Based Routing Protocols (AODV, DSR and DSDV)," *International Journal for Advance Researh in Engineering and Technology*, vol. 1, pp. 54-58, 2013.

[13] C. C. a. A. Molinaro, "Improving V2R connectivity to provide ITS application in IEEE 802.11p/WAVE VANETs," 2012.

[14] nsnam, "About NS-3," [Online]. Available: https://www.nsnam.org/docs/release/3.21/tutorial/html/introduction.html#about-ns3.

[15] nsnam, "About NS-3," 5 December 2015. [Online]. Available: https://www.nsnam.org/docs/release/3.21/tutorial/html/introduction.html#about-ns3. [Diakses 2 December 2016].

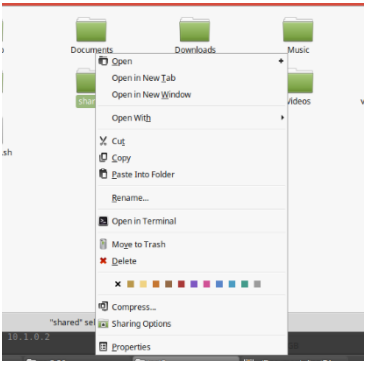
[16] nsnam, "Ad Hoc On-Demand Distance Vector," ns nam, 19 Desember 2014. [Online]. Available: http://www.nsnam.org/docs/models/html/aodv.html. [Diakses 12 Desember 2016].

[17] S. M. Kumbura, "Slideshare," Slideshare, [Online]. Available: https://www.slideshare.net/shankamahakumbura/aodv-protocol-34793714. [Diakses 22 May 2017].

LAMPIRAN

Lampiran 1. Implementasi *shared folder*

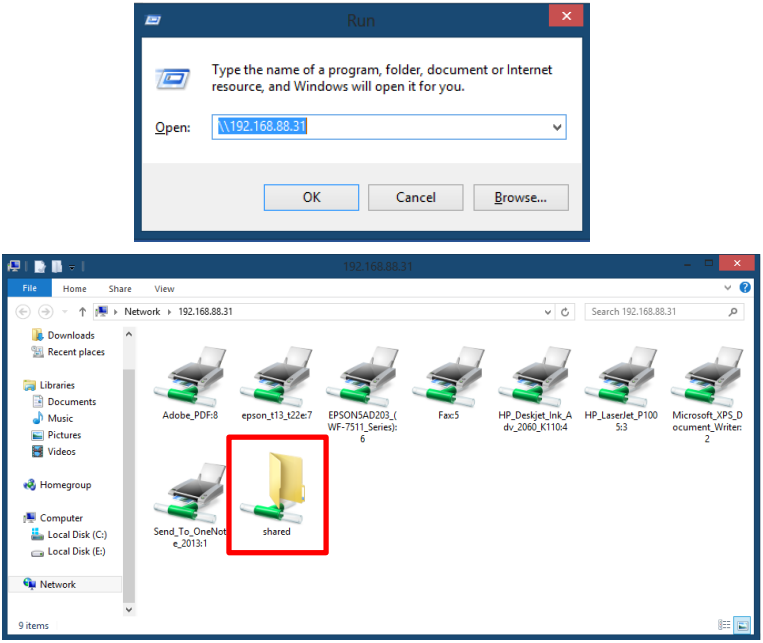
Untuk memudahkan mobilisasi data dari VMware work station ke windows diperlukan sebuah shared folder. Pertama yang harus dilakukan adalah membuat *folder* bernama *shared* pada Linux. Kemudian klik kanan *folder* tersebut dan pilih *sharing option* seperti pada gambar dibawah ini.



Kemudian akan muncul *popup* menu *Folder Sharing*. Tandai ketiga opsi yang terdapat pada pilihan menu tersebut seperti pada gambar dibawah ini.

Folder bernama *shared* telah berhasil menjadi perantara antara Windows dan Linux. Untuk mengakses folder tersebut gunakan Win+R kemudian ketikkan alamat IP dari Linux. Pastikan

sebelumnya ketika instalasi Linux sebagai *guest*, *network mode*-nya merupakan *bridged adapter* yang berarti komputer *host* dan *guest* berada pada satu *subnet*. Untuk cara mengakses *folder sharing* tersebut dapat dilihat pada gambar di bawah ini.



Lampiran 2. Petunjuk Instalasi NS-3

Untuk instalasi NS-3, yang pertama kali dilakukan yaitu instalasi dependensi yang akan dibutuhkan selama menggunakan NS-3. Terdapat banyak depedensi yang diperlukan oleh NS-3 oleh karena itu untuk mencegah *error* dan supaya aman dan stabil maka alangkah baiknya jika semua depedensi yang diperlukan telah terpasang. Perintah untuk memasang depedensi yang diperlukan ditunjukkan oleh perintah di bawah ini.

```
$ sudo apt-get install gcc g++ python python-dev bzr
mercurial gdb valgrind gsl-bin libgsl0-dev
libgsl0ldbl flex bison libfl-dev tcpdump sqlite
sqlite3 libsqlite3-dev libxml2 libxml2-dev
libgtk2.0-0 libgtk2.0-dev vtun lxc uncrustify
doxygen graphviz imagemagick texlive texlive-extra-
utils texlive-latex-extra python-sphinx dia python-
pygraphviz python-kiwi python-pygoocanvas
libgoocanvas-dev libboost-signals-dev libboost-
filesystem-dev
```

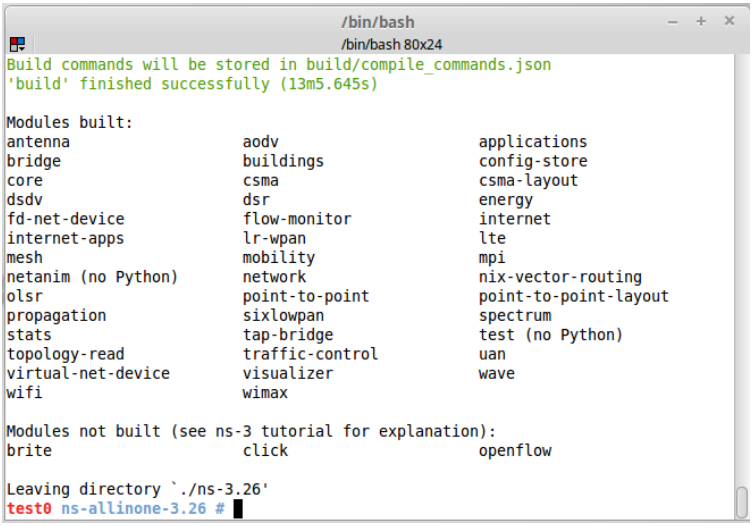
Setelah semua depedensi lengkap, dilakukan unduh modul NS-3 menggunakan Tarball dan ekstraksi *file* modul dengan perintah di bawah ini.

```
$ cd
$ mkdir workspace
$ wget https://www.nsnam.org/release/ns-allinone-
3.26.tar.bz2
$ tar xjf ns-allinone-3.26.tar.bz2
```

File tarbal NS-3 yang telah selesai diunduh kemudian dilakukan proses *build* dengan program build.py yang ada pada direktori ns-allinone-3.26. Program ini akan mengkonfigurasi NS-3 sesuai dengan keperluan pengguna pada umumnya. Untuk melakukan *build* gunakan perintah di bawah ini.

```
$ ./build.py --enable-examples --enable-tests
```

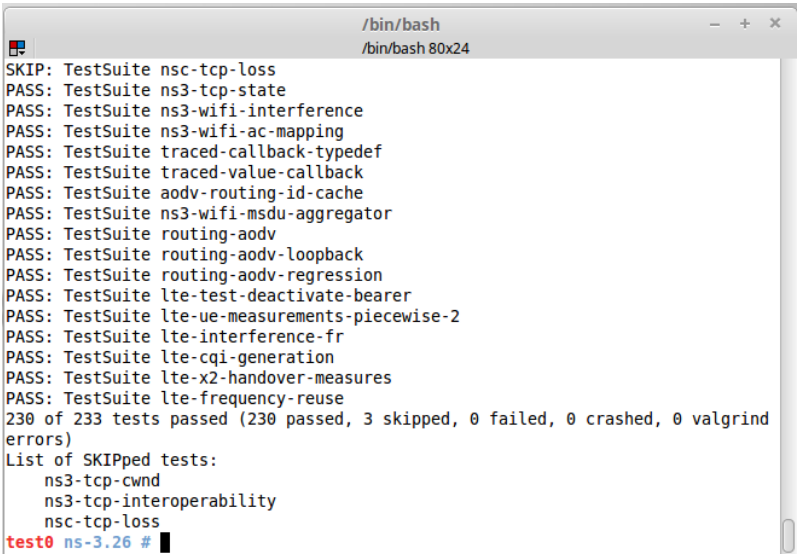
Oleh karena pada Tugas Akhir ini penulis membutuhkan modul *example* dan *test*, maka dari itu disisipkan pada argumen untuk build.py. Hasil setelah melakukan *build* menggunakan build.py dapat dilihat pada gambar di bawah ini.



Proses *build* pertama kalinya akan memakan waktu yang cukup lama. Pastikan hasil output setelah melakukan build pada NS-3 sudah benar. Yakni semua modul berhasil kecuali *brite*, *click* dan *openflow*. Setelah *build* berhasil, lakukan testing terhadap fungsi-fungsi NS-3. Untuk melakukan testing gunakan perintah seperti di bawah ini.

```
$ ./test.py -c core
```

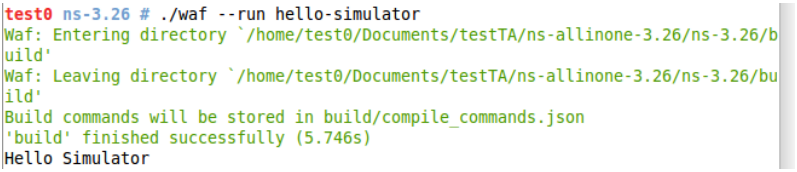
Tes tersebut berjalan secara parallel oleh *Waf* dan hasilnya memunculkan status per tes yang dilakukan dan pada bagian akhir terdapat *report* seperti di bawah ini. Pengetesan fungsi-fungsi NS-3 ini juga memakan waktu yang lama. Terdapat 3 tes yang dilewatkan atau berstatus *SKIPPED* namun itu tidak masalah.



Untuk percobaan *running* salah satu program NS-3, dapat dilakukan dengan menjalankan contoh program *hello-simulator* dengan perintah seperti di bawah ini.

```
$ ./waf –run hello-simulator
```

Hasil dari program hello-simulator dapat dilihat pada Gambar di bawah ini.



Lampiran 3. Script awk untuk menghitung PDR

```
BEGIN {
	sent =0;
	recv =0;
	recv_id=0;
	pdr=0;
}
{
	#count packet transmit
	if(	$1	==	"t"	&&	$3	==
"/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/S
tate/Tx" && $9 == "Retry=0," && $50 == "(size=64)"){
		sent++;
	}
	#count packet receive
	if(	$1	==	"r"	&&	$3	==
"/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/Phy/S
tate/RxOk" && $50 == "(size=64)"){
		recv++;
	}
}
END {
	pdr = ( recv/sent )*100;
	print "Transmitted packet(s) = ", sent;
	print "Received packet(s) = ", recv;
	print "Packet Delivery Ratio = ", pdr, "%";
}
```

Lampiran 4. Script awk untuk menghitung Delay

```
BEGIN {
	for ( i in pkt_id) {
		pkt_id[i] = 0;
	}

	for ( i in pkt_sent) {
		pkt_sent[i] = 0;
	}

	for ( i in pkt_recv ) {
		pkt_recv[i] = 0;
	}
```

```
	delay = avg_delay = 0;
	recv = 0;
}
{
	# store packet send time
	if	(	$1	==	"t"	&&	$3	==
"/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/S
tate/Tx" && $9 == "Retry=0," && $50 == "(size=64)" )
{
		pkt_sent[$30] = $2;
	}
	# store packet receive time
	if	(	$1	==	"r"	&&	$3	==
"/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/Phy/S
tate/RxOk" && $50 == "(size=64)" ) {
		recv++;
		pkt_recv[$30] = $2;
	}
}
END {
for (i in pkt_recv) {
	delay += pkt_recv[i] - pkt_sent[i];
	num ++;
}
avg_delay = delay / num;
print "Average Packet Delivery Delay = ", avg_delay,
"second";
}
```

Lampiran 5. Script awk untuk menghitung Routing Overhead

```
BEGIN {
	rreq=0;
	rrep=0;
	rerr=0;
	total=0;
}
{
	if ( $1 == "t" && $50 == "(RREQ)" && $57 ==
"10.1.0.1" && $63 == "10.1.0.2") {
		rreq++;
	}
```

```
       if ( $1 == "t" && $50 == "(RREP)" && $60 ==
"10.1.0.1" && $54 == "10.1.0.2") {
              rrep++;
       }
       if ( $1 == "t" && $50 == "(RRER)" && $58 ==
"10.1.0.1)" ) {
        rerr++;
    }
}
END {
       total = rreq + rrep + rerr;
       print "Jumlah RREQ = ", rreq;
       print "Jumlah RREP = ", rrep;
       print "Jumlah RERR = ", rerr;
       print "Jumlah RO   = ", total;
}
```

Lampiran 6. Potongan kode untuk menjalankan routing protocol sesuai dengan skenario yang telah di set.

```
void
VanetRoutingExperiment::SetupScenario ()
{
  // member variable parameter use
  // defaults or command line overrides,
  // except where scenario={1,2,3,...}
  // have been specified, in which case
  // specify parameters are overwritten
  // here to setup for specific scenarios

  // certain parameters may be further overridden
  // i.e. specify a scenario, override tx power.

  if (m_scenario == 1)
    {
      // 40 nodes in RWP 300 m x 1500 m synthetic
highway, 10s
      m_traceFile = "";
      m_logFile = "";
      m_mobility = 2;
      if (m_nNodes == 156)
        {
          m_nNodes = 40;
        }
```

```
      if (m_TotalSimTime == 300.01)
        {
          m_TotalSimTime = 10.0;
        }
    }
  else if (m_scenario == 2)
    {
      // Realistic vehicular trace in 4.6 km x 3.0 km
suburban Zurich
      // "low density, 99 total vehicles"
      m_traceFile   =   "src/wave/examples/low99-ct-
unterstrass-1day.filt.7.adj.mob";
      m_logFile       =       "low99-ct-unterstrass-
1day.filt.7.adj.log";
      m_mobility = 1;
      m_nNodes = 99;
      m_TotalSimTime = 300.01;
      m_nodeSpeed = 0;
      m_nodePause = 0;
      m_CSVfileName       =       "low_vanet-routing-
compare.csv";
      m_CSVfileName       =       "low_vanet-routing-
compare2.csv";
    }
  else if (m_scenario == 3)
    {
        m_traceFile = "scratch/20scene1.tcl";
        m_trName = "20scene1";
        m_routingTables = 1;
        m_nNodes = 50;
        m_TotalSimTime = 200;
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_txp = 7;
    }
  else if (m_scenario == 4)
    {
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_routingTables = 1;
        m_traceFile = "scratch/20scene2.tcl";
        m_trName = "20scene2";
        m_nNodes = 50;
        m_txp = 7;
```

```
        m_TotalSimTime = 200;
    }
  else if (m_scenario == 5)
    {
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_routingTables = 1;
        m_traceFile = "scratch/20scene3.tcl";
        m_trName = "20scene3";
        m_nNodes = 50;
        m_txp = 7;
        m_TotalSimTime = 200;
    }
  else if (m_scenario == 6)
    {
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_routingTables = 1;
        m_traceFile = "scratch/20scene4.tcl";
        m_trName = "20scene4";
        m_nNodes = 50;
        m_txp = 7;
        m_TotalSimTime = 200;
    }
  else if (m_scenario == 7)
    {
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_routingTables = 1;
        m_traceFile = "scratch/20scene5.tcl";
        m_trName = "20scene5";
        m_nNodes = 50;
        m_txp = 7;
        m_TotalSimTime = 200;
    }
  else if (m_scenario == 8)
    {
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_routingTables = 1;
        m_traceFile = "scratch/20scene6.tcl";
        m_trName = "20scene6";
        m_nNodes = 50;
        m_txp = 7;
```

```
        m_TotalSimTime = 200;
    }
  else if (m_scenario == 9)
    {
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_routingTables = 1;
        m_traceFile = "scratch/20scene7.tcl";
        m_trName = "20scene7";
        m_nNodes = 50;
        m_txp = 7;
        m_TotalSimTime = 200;
    }
  else if (m_scenario == 10)
    {
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_routingTables = 1;
        m_traceFile = "scratch/20scene8.tcl";
        m_trName = "20scene8";
        m_nNodes = 50;
        m_txp = 7;
        m_TotalSimTime = 200;
    }
  else if (m_scenario == 11)
    {
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_routingTables = 1;
        m_traceFile = "scratch/20scene9.tcl";
        m_trName = "20scene9";
        m_nNodes = 50;
        m_txp = 7;
        m_TotalSimTime = 200;
    }
  else if (m_scenario == 12)
    {
        m_asciiTrace = 1;
        m_nSinks = 1;
        m_routingTables = 1;
        m_traceFile = "scratch/20scene10.tcl";
        m_trName = "20scene10";
        m_nNodes = 50;
        m_txp = 7;
```

```
        m_TotalSimTime = 200;
    }

}
```

Lampiran 7. Potongan Kode perubahan pada fungsi ProcessHello di aodv-routing-protocol.cc

```
void
RoutingProtocol::ProcessHello (RrepHeader const &
rrepHeader, Ipv4Address receiver )
{
  NS_LOG_FUNCTION (this << "from " <<
rrepHeader.GetDst ());
  /*
   *  Whenever a node receives a Hello message from
a neighbor, the node
   * SHOULD make sure that it has an active route to
the neighbor, and
   * create one if necessary.
   */
  RoutingTableEntry toNeighbor;
  if (!m_routingTable.LookupRoute (rrepHeader.GetDst
(), toNeighbor)) //if no route to new neighbor
    {
      Ptr<NetDevice> dev = m_ipv4->GetNetDevice
(m_ipv4->GetInterfaceForAddress (receiver));
      RoutingTableEntry newEntry (/*device=*/ dev,
/*dst=*/ rrepHeader.GetDst (), /*validSeqNo=*/ true,
/*seqno=*/ rrepHeader.GetDstSeqno (),

/*iface=*/        m_ipv4->GetAddress        (m_ipv4-
>GetInterfaceForAddress (receiver), 0),

/*hop=*/  1,  /*nextHop=*/  rrepHeader.GetDst  (),
/*lifeTime=*/ rrepHeader.GetLifeTime (),

/*added*/rrepHeader.GetSpeed(),
rrepHeader.GetAccel(),            rrepHeader.GetX(),
rrepHeader.GetY());
      m_routingTable.AddRoute (newEntry);
      m_ipv4AddressEntry.insert(std::make_pair
(rrepHeader.GetDst(),newEntry)); //added
    }
```

```
  else
    {
      //if there's a route
      toNeighbor.SetLifeTime     (std::max     (Time
(m_allowedHelloLoss       *       m_helloInterval),
toNeighbor.GetLifeTime ()));
      toNeighbor.SetSeqNo    (rrepHeader.GetDstSeqno
());
      toNeighbor.SetValidSeqNo (true);
      toNeighbor.SetFlag (VALID);
      toNeighbor.SetOutputDevice            (m_ipv4-
>GetNetDevice        (m_ipv4->GetInterfaceForAddress
(receiver)));
      toNeighbor.SetInterface    (m_ipv4->GetAddress
(m_ipv4->GetInterfaceForAddress (receiver), 0));
      toNeighbor.SetHop (1);
      toNeighbor.SetNextHop (rrepHeader.GetDst ());
      //added
      toNeighbor.SetSpeed (rrepHeader.GetSpeed());
      toNeighbor.SetAccel (rrepHeader.GetAccel());
      toNeighbor.SetX (rrepHeader.GetX());
      toNeighbor.SetY (rrepHeader.GetY());
      m_routingTable.Update (toNeighbor);

      m_ipv4AddressEntry.insert(std::make_pair
(rrepHeader.GetDst(),toNeighbor)); //added
    }

  if (m_enableHello)
    {
      m_nb.Update    (rrepHeader.GetDst    (),    Time
(m_allowedHelloLoss * m_helloInterval));
    }

  TWR_total=0, nt=0;

  for                          (std::map<Ipv4Address,
RoutingTableEntry>::const_iterator        j        =
m_ipv4AddressEntry.begin       ();       j       !=
m_ipv4AddressEntry.end (); ++j){
    Ipv4Address addr = j->first;
    RoutingTableEntry rte = j->second;

    //check if neighbor or not and calculate TWR
```

```
    if (m_nb.IsNeighbor(addr)){
      nt++;
      dx = m_x - rte.GetX();
      dy = m_y - rte.GetY();
      dis = sqrt((dx*dx)+(dy*dy));
      modSpeed = fs * abs(m_speed - rte.GetSpeed());
      modAccel = fa * abs(m_accel - rte.GetAccel());
      modDistance = fd * dis;
      TWR_total = modSpeed + modAccel + modDistance;
    }
  }
  TWR_total = TWR_total/nt;

  //get new interval value
  if (TWR_total >= TWRmax) {
    intervalNow = Imin;
  } else if (TWRmin <= TWR_total && TWR_total <=
TWRmax) {
    intervalNow = (TWRmax/TWR_total)*Imin;
  } else if (TWR_total <= TWRmin){
    intervalNow = Imax;
  }

  m_helloInterval = Seconds(intervalNow);
}
```

Lampiran 8. Potongan kode yang dimodifikasi pada fungsi SendHello

```
void
RoutingProtocol::SendHello ()
{
  NS_LOG_FUNCTION (this);
  /* Broadcast a RREP with TTL = 1 with the RREP
message fields set as follows:
   *   Destination IP Address         The node's IP
address.
   *    Destination Sequence Number      The node's
latest sequence number.
   *   Hop Count                      0
   *   Lifetime                     AllowedHelloLoss
* HelloInterval
   */
   //added field
```

```
  for                          (std::map<Ptr<Socket>,
Ipv4InterfaceAddress>::const_iterator      j      =
m_socketAddresses.begin       ();       j       !=
m_socketAddresses.end (); ++j)
    {
      Ptr<Socket> socket = j->first; //ragu
      Ipv4InterfaceAddress iface = j->second;

      //get velocity and position from this node
      Ptr<Node> thisNode = socket->GetNode();
      Ptr<MobilityModel>  thisMobility  =  thisNode-
>GetObject<MobilityModel>();
      Vector    thisVelocity    =    thisMobility-
>GetVelocity();
      Vector    thisPosition    =    thisMobility-
>GetPosition();
      double                  speed                =
sqrt((thisVelocity.x*thisVelocity.x)+(thisVelocity.y
*thisVelocity.y));

      //assign acelleration and update another field
      Time now = Simulator::Now();
      if(now - m_now != 0)
      {
        m_accel    =    (speed    -    m_speed)    /
(now.GetSeconds() - m_now);
        //update position, speed, acelleration
        m_x = thisPosition.x;
        m_y = thisPosition.y;
        m_speed = speed;
        m_now   =   now.GetSeconds();   //update   last
update time
      }
      m_speed2 = modf(m_speed, &m_speed1);
      m_speed2 = m_speed2*1000000;
      m_accel2 = modf(m_accel, &m_accel1);
      m_accel2 = m_accel2*1000000;
      m_x2 = modf(m_x, &m_x1);
      m_x2 = m_x2*1000000;
      m_y2 = modf(m_y, &m_y1);
      m_y2 = m_y2*1000000;
```

```
      RrepHeader helloHeader (/*prefix size=*/ 0,
/*hops=*/ 0, /*dst=*/ iface.GetLocal (), /*dst
seqno=*/ m_seqNo,

/*origin=*/ iface.GetLocal (),/*lifetime=*/ Time
(m_allowedHelloLoss * m_helloInterval),

/*speed*/ m_speed1, m_speed2, /*acceleration*/
m_accel1, m_accel2, /*x*/ m_x1, m_x2, /*y*/ m_y1,
m_y2);
      Ptr<Packet> packet = Create<Packet> ();
      SocketIpTtlTag tag;
      tag.SetTtl (1);
      packet->AddPacketTag (tag);
      packet->AddHeader (helloHeader);
      TypeHeader tHeader (AODVTYPE_RREP);
      packet->AddHeader (tHeader);
      // Send to all-hosts broadcast if on /32 addr,
subnet-directed otherwise
      Ipv4Address destination;
      if (iface.GetMask () == Ipv4Mask::GetOnes ())
        {
          destination = Ipv4Address
("255.255.255.255");
        }
      else
        {
          destination = iface.GetBroadcast ();
        }
      Time jitter = Time (MilliSeconds
(m_uniformRandomVariable->GetInteger (0, 10)));
      Simulator::Schedule (jitter,
&RoutingProtocol::SendTo, this , socket, packet,
destination);
    }
}
```

Lampiran 9. Potongan kode fungsi yang di modifikasi pada aodv-packet.cc

```
void
RrepHeader::Serialize (Buffer::Iterator i) const
{
  i.WriteU8 (m_flags);
```

```
  i.WriteU8 (m_prefixSize);
  i.WriteU8 (m_hopCount);
  WriteTo (i, m_dst);
  i.WriteHtonU32 (m_dstSeqNo);
  WriteTo (i, m_origin);
  i.WriteHtonU32 (m_lifeTime);

  //added field
  i.WriteU32 ((unsigned int)m_speed1);
  i.WriteU32 ((unsigned int)m_speed2);
  i.WriteU32 ((unsigned int)m_accel1);
  i.WriteU32 ((unsigned int)m_accel2);
  i.WriteU32 ((unsigned int)m_x1);
  i.WriteU32 ((unsigned int)m_x2);
  i.WriteU32 ((unsigned int)m_y1);
  i.WriteU32 ((unsigned int)m_y2);
}

uint32_t
RrepHeader::Deserialize (Buffer::Iterator start)
{
  Buffer::Iterator i = start;

  m_flags = i.ReadU8 ();
  m_prefixSize = i.ReadU8 ();
  m_hopCount = i.ReadU8 ();
  ReadFrom (i, m_dst);
  m_dstSeqNo = i.ReadNtohU32 ();
  ReadFrom (i, m_origin);
  m_lifeTime = i.ReadNtohU32 ();

  //added field

  m_speed1 = (double)i.ReadU32();
  m_speed2 = (double)i.ReadU32();
  if (m_speed1<1000000 && m_speed2<1000000){
    m_speed = m_speed1+(m_speed2/1000000);
  } else if (m_speed1>1000000 && m_speed1<4294967296
&& m_speed2>1000000 && m_speed2<4294967296){
    m_speed  =  (m_speed1-4294967296)  +  ((m_speed2-
4294967296)/1000000);
  }

  m_accel1 = (double)i.ReadU32();
```

```
  m_accel2 = (double)i.ReadU32();
  if (m_accel1<1000000 && m_accel2<1000000){
    m_accel = m_accel1+(m_accel2/1000000);
  } else if (m_accel1>1000000 && m_accel1<4294967296
&& m_accel2>1000000 && m_accel2<4294967296){
    m_accel = (m_accel1-4294967296) + ((m_accel2-
4294967296)/1000000);
  }

  m_x1 = (double)i.ReadU32();
  m_x2 = (double)i.ReadU32();
  if (m_x1<1000000 && m_x2<1000000){
    m_x = m_x1+(m_x2/1000000);
  } else if (m_x1>1000000 && m_x1<4294967296 &&
m_x2>1000000 && m_x2<4294967296){
    m_x = (m_x1-4294967296) + ((m_x2-
4294967296)/1000000);
  }

  m_y1 = (double)i.ReadU32();
  m_y2 = (double)i.ReadU32();
  if (m_y1<1000000 && m_y2<1000000){
    m_y = m_y1+(m_y2/1000000);
  } else if (m_y1>1000000 && m_y1<4294967296 &&
m_y2>1000000 && m_y2<4294967296){
    m_y = (m_y1-4294967296) + ((m_y2-
4294967296)/1000000);
  }

  uint32_t dist = i.GetDistanceFrom (start);
  NS_ASSERT (dist == GetSerializedSize ());
  return dist;
}
```

Lampiran 10. Potongan kode penambahan fungsi dan atribut pada aodv-rtable.h

```
class RoutingTableEntry
{
public:
  /// c-to
  RoutingTableEntry (Ptr<NetDevice> dev =
0,Ipv4Address dst = Ipv4Address (), bool vSeqNo =
false, uint32_t m_seqNo = 0,
```

```
                      Ipv4InterfaceAddress   iface   =
Ipv4InterfaceAddress (), uint16_t  hops = 0,
                      Ipv4Address      nextHop      =
Ipv4Address (), Time lifetime = Simulator::Now (),
/*added field*/ double speed = 0, double accel = 0,
double x = 0, double y = 0);

  ~RoutingTableEntry ();

  ///\name Precursors management
  //\{
  /**
   * Insert precursor in precursor list if it doesn't
yet exist in the list
   * \param id precursor address
   * \return true on success
   */
  bool InsertPrecursor (Ipv4Address id);
  /**
   * Lookup precursor by address
   * \param id precursor address
   * \return true on success
   */
  bool LookupPrecursor (Ipv4Address id);
  /**
   * \brief Delete precursor
   * \param id precursor address
   * \return true on success
   */
  bool DeletePrecursor (Ipv4Address id);
  /// Delete all precursors
  void DeleteAllPrecursors ();
  /**
   * Check that precursor list empty
   * \return true if precursor list empty
   */
  bool IsPrecursorListEmpty () const;
  /**
   * Inserts precursors in vector prec if they does
not yet exist in vector
   */
  void  GetPrecursors  (std::vector<Ipv4Address>  &
prec) const;
  //\}
```

```
  /// Mark entry as "down" (i.e. disable it)
  void Invalidate (Time badLinkLifetime);

  // Fields
  Ipv4Address GetDestination () const { return
m_ipv4Route->GetDestination (); }
  Ptr<Ipv4Route> GetRoute () const { return
m_ipv4Route; }
  void SetRoute (Ptr<Ipv4Route> r) { m_ipv4Route = r;
}
  void SetNextHop (Ipv4Address nextHop) {
m_ipv4Route->SetGateway (nextHop); }
  Ipv4Address GetNextHop () const { return
m_ipv4Route->GetGateway (); }
  void SetOutputDevice (Ptr<NetDevice> dev) {
m_ipv4Route->SetOutputDevice (dev); }
  Ptr<NetDevice> GetOutputDevice () const { return
m_ipv4Route->GetOutputDevice (); }
  Ipv4InterfaceAddress GetInterface () const { return
m_iface; }
  void SetInterface (Ipv4InterfaceAddress iface) {
m_iface = iface; }
  void SetValidSeqNo (bool s) { m_validSeqNo = s; }
  bool GetValidSeqNo () const { return m_validSeqNo;
}
  void SetSeqNo (uint32_t sn) { m_seqNo = sn; }
  uint32_t GetSeqNo () const { return m_seqNo; }
  void SetHop (uint16_t hop) { m_hops = hop; }
  uint16_t GetHop () const { return m_hops; }
  void SetLifeTime (Time lt) { m_lifeTime = lt +
Simulator::Now (); }
  Time GetLifeTime () const { return m_lifeTime -
Simulator::Now (); }
  void SetFlag (RouteFlags flag) { m_flag = flag; }
  RouteFlags GetFlag () const { return m_flag; }
  void SetRreqCnt (uint8_t n) { m_reqCount = n; }
  uint8_t GetRreqCnt () const { return m_reqCount; }
  void IncrementRreqCnt () { m_reqCount++; }
  void SetUnidirectional (bool u) { m_blackListState
= u; }
  bool IsUnidirectional () const { return
m_blackListState; }
```

```
  void     SetBalcklistTimeout     (Time     t)     {
m_blackListTimeout = t; }
  Time   GetBlacklistTimeout   ()   const   {   return
m_blackListTimeout; }

  //added field
  void SetSpeed (double s) { m_speed = s; };
  double GetSpeed () const { return m_speed; };
  void SetAccel (double a) { m_accel = a; };
  double GetAccel () const { return m_accel;};
  void SetX (double x) { m_x = x;};
  double GetX () const { return m_x; };
  void SetY (double y) { m_y = y;};
  double GetY () const { return m_y; };

  /// RREP_ACK timer
  Timer m_ackTimer;

  /**
   * \brief Compare destination address
   * \return true if equal
   */
  bool operator== (Ipv4Address const  dst) const
  {
    return (m_ipv4Route->GetDestination () == dst);
  }
  void Print (Ptr<OutputStreamWrapper> stream) const;

private:
  /// Valid Destination Sequence Number flag
  bool m_validSeqNo;
  /// Destination Sequence Number, if m_validSeqNo =
true
  uint32_t m_seqNo;
  /// Hop Count (number of hops needed to reach
destination)
  uint16_t m_hops;
  /**
  * \brief Expiration or deletion time of the route
  *    Lifetime field in the routing table plays dual
role --
  *    for an active route it is the expiration time,
and for an invalid route
  *    it is the deletion time.
```

```
   */
   Time m_lifeTime;
   /** Ip route, include
   *    - destination address
   *    - source address
   *    - next hop address (gateway)
   *    - output device
   */
   Ptr<Ipv4Route> m_ipv4Route;
   /// Output interface address
   Ipv4InterfaceAddress m_iface;
   /// Routing flags: valid, invalid or in search
   RouteFlags m_flag;

   /// List of precursors
   std::vector<Ipv4Address> m_precursorList;
   /// When I can send another request
   Time m_routeRequestTimout;
   /// Number of route requests
   uint8_t m_reqCount;
   /// Indicate if this entry is in "blacklist"
   bool m_blackListState;
   ///  Time  for  which  the  node  is  put  into  the
blacklist
   Time m_blackListTimeout;

   //added field
   double m_speed;
   double m_accel;
   double m_x;
   double m_y;
};
```

BIODATA PENULIS

I Dewa Putu Sumitra Putra, lahir di Ibukota Jakarta, 16 Juni 1995. Penulis adalah anak pertama dari tiga bersaudara. Menempuh pendidikan di SD No. 1 Ubung, SMP Negeri 10 Denpasar, SMA Negeri 4 Denpasar, dan terakhir melanjutkan kuliah di jurusan Teknik Informatika – ITS.

Selama berkuliah penulis aktif dalam kegiatan dan organisasi himpunan mahasiswa informatika sebagai Kepala Bidang Media Kreatif Departemen Media dan Informasi Himpunan Mahasiswa Teknik Computer-Informatika.

Selain itu, penulis juga aktif dalam organisasi kampus Tim Pembina Kerohanian Hindu-ITS sebagai Kepala Departemen Komunikasi dan Informasi.

Selain menjalankan tugas mahasiswa, penulis juga aktif menjadi asisten dosen pada PIKTI-ITS.

Ketertarikan penulis dibidang informatika berada pada bidang sistem teknologi informasi, sekuritas jaringan, perancangan keamanan sistem dan jaringan, teknologi antar jaringan, dan teknologi tepat guna.

Penulis dapat dihubungi dengan mengirimkan pesan elektronik ke alamat desumputra@gmail.com.